社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2007－DBS－143（61）
2007／7／3

コスト見積に基づくデータベースの再構造化支援

馬強† 石黒義英†

† NECサービスプラットフォーム研究所 〒630-0101 奈良県生駒市高山町8916-47

あらまし データベースのスキーマ構成は，サービスやアプリケーションの設計時に，静的に定義・最適化されることが多い．しかしながら，アプリケーションの増減や業務変更などによりデータの利用形態が変化することで，データ共有・アクセスの効率が悪化する場合がある．このような場合，データ特性，リソース制約および利用要求に応じた動的なデータベースの再構造化は重要な解決手段となる．しかしながら，データベースの再構造化は，人手に頼ることが多く，データベース設計者や管理者にとって非常に負担の大きな作業となっている．そこで，本稿では，データベースの再構造化作業の効率化を目的として，データベースのアクセスと蓄積の両面からコストを見積る手法と，その結果の可視化手法を提案し，提案したコスト見積手法の評価と考察を行う．

キーワード データベース再構造化，コスト見積，アクセスコスト，蓄積コスト

Database Restructing Support Based on Cost Estimation

Qiang MA† and Yoshihide ISHIGURO†

† Takayama-cho 8916-47, Ikoma, Nara 6300101, Japan

Abstract Generally, the database schemata of a certain appliaction or service are designed in advance. However, since data increase rapidly and its usage and access pattern change dynamically, the database perfomance will go down in some cases. Dynamic restructing of database schemata is one of the solutions for this issue. However, the conventional methods of database restructing just try to create or modify views based on the access-cost estimation. They do not support advanced database restructing functions, such as schema merger and division, and database restructing based on storage-cost estimation. Hence, database restructing relies on database experts and is a very expensive task yet. In this paper, we propose methods of database cost-estimation and visualization for supporting database restructing. Based on these methods, we can compare the costs of different database schemata without changing the schemata in a RDB system. We also introduce a prototype system under developing and some evaluation results.

Key words Database Restructing, Cost Estimation, Access Cost, Storage Cost

1. はじめに

近年，複数のサービスやアプリケーションをホスティングして，多くのサービスを同時に提供する基盤（例えば，SaaS）などが注目されている．一般的に，各々のサービスやアプリケーションで扱うデータベースは，サービスまたはアプリケーションの設計時に静的に定義・最適化される．しかしながら，各々のサービスでデータベースの最適化を図っても，データのシェアリングを前提とする基盤全体のデータベースの最適化には繋がらない．そこで利用されるデータの量と種類は膨大となり，それらを静的にかつ個別に管理しているとデータ処理および蓄積の効率が悪くなる．これは、サービス提供のスケーラビリティという点で問題となる．そのため，データ特性，リソース制約および利用要求に応じた動的なデータベース最適化技術が有効である．

データベースの最適化技術は，文献[1][2]に詳しい．データベースの最適化技術は，データベースの再構造化，質問最適化，索引の最適化、データ配置の最適化など様々な手法がある．ここでは，データベースの再構造化に着目する．データベースの再構造化とは，データのアクセスと蓄積の効率を向上させるため，データベースにおけるデータのあり方（データベースの構成）を変更することである．このようなデータベースの再構造化は，近年の企業合併・買収に伴う情報システムの統合にも重要である．本論文では，データベースにおけるデータのあり方を，データのスキーマ，スキーマの種類と数として定義し，スキーマ集合を用いて表現する．

データベースの再構造化手法は，従来から多数存在している[3]～[6]．これらの手法では，データベース上のアクセスログを解析し，アクセスコストに基づいて索引やビュー（マテリアライズドビュー，索引付きビューなど）を自動生成・推薦する

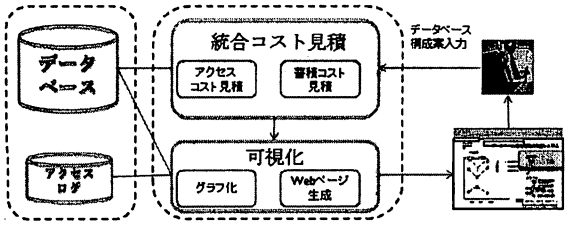

図 1 DB 再構造化支援システムの構成図

ことでデータのアクセス効率の向上を図っている．つまり，新しいスキーマ（ビュー）の作成に基づくデータベースの再構造化を行っている．しかしながら，これらの手法では，以下のような課題があり，より高度なデータベース再構造化の設計は人手に頼らざる得ないため，データベース設計者にとって負担の大きい作業である．

- 従来のデータベース再構造化の支援手法では，ビューの新規作成しかサポートせず，テーブルの分割（スキーマの分割）やマージ（スキーマの結合）といった高度な操作によるデータベースの再構造化は人手に頼らざる得ない．
- 従来のデータベース再構造化の支援手法では，蓄積容量とアクセス時間が同時に考慮されていなかったため，パフォーマンス（アクセス速度,etc.）と保守コスト（蓄積容量, etc.）の関係が把握しにくく，リソースの制限，利用要求やデータ特性に応じた柔軟なデータベース構成の設計が困難な場合がある．

そこで，我々は，データベースのアクセスログ，データベースとファイルシステムのカタログ情報を利用した，データベースのアクセスと蓄積コストの統合見積手法と，その結果の可視化手法を提案し，データベースの再構造化の設計支援を試みる．実際のデータベースを変更することなしに，様々な構成案でのコストを直感的に比較できるため，データベースの設計者の負担の軽減が期待できる．

以下，本稿の構成を示す．2. 節では，提案手法を用いたデータベース再構造化支援システムについて述べる．3. 節と 4. 節では，それぞれ，コストの見積とその可視化について説明する．5. 節では，提案手法の評価実験について述べる．最後に，まとめと今後の課題について 6. 節で述べる．

2. データベース再構造化支援システム

データベースコストの見積と可視化手法を用いて，我々は，データベース再構造化支援システムの研究開発を進めている．そのプロトタイプシステムの構成を図 1 に示す．ユーザがデータベースの構成案を入力すると，統合コスト見積モジュールが，蓄積コスト見積モジュールとアクセスコスト見積モジュールに，それぞれコストの見積を依頼し，その結果を統合してデータベースの統合コストを計算する．可視化モジュールは，これらのコスト（蓄積コスト，アクセスコスト，統合コスト）をグラフにプロットして，データベースとスキーマレベルのコストグラフを生成する．さらに，可視化モジュールは，Web ページを生成し，コストグラフおよびデータベースとスキーマの構成情報を表示して，ユーザに提示する．ユーザがこれらの Web ページを閲覧することによって，異なる構成のデータベースのコストを比較できる．さらに，新たなデータベース構成を入力して，データベース構成をさらに改良することも可能である．

表 1 データベースとファイルシステムのカタログ情報

	項目	説明
テーブル r	$T(r)$	r に格納されているタプル数
	h	タプルのヘッダサイズ
	A_r	r の属性集合
	$size(a_i)$	属性 $a_i \in A_r$ のサイズ
	$V(r, a_i)$	属性 a_i の r 上の値の種類数
ファイルシステム	b	ブロックのサイズ
	bh	ブロックヘッダのサイズ

3. コスト見積手法

3.1 スキーマの統合操作

本稿では，関係データベースを対象に，スキーマをテーブルとビューに分類し，属性と条件の集合で表す．関係データベースの再構造化では，スキーマ操作を通して，データベースのスキーマ集合が更新される．本研究では，データベース再構造化のスキーマ操作を，結合，分割，作成・削除，および種類変換の 4 種類に分類して定義する．以下，特別の説明がなければ，s, s_1, s_2 を新規スキーマとし，r, r_1, r_2 を既存スキーマとする．

- 結合 ($s = r_1 \oplus r_2$): 二つ以上のスキーマ (r_1, r_2) を一つのスキーマ (s) にマージする操作である．
- 分割 ($s_1 \oplus s_2 = r$): 一つのスキーマ (r) を二つ以上のスキーマ (s_1, s_2) に分割する操作である．
- 作成・削除 ($s = q(R)$) : 作成は既存のスキーマ群 (R) から新しいスキーマ (s) を生成する操作である．削除は，既存スキーマの削除操作である．
- 種類変換 ($s \sim r$) : ビュー (r) をテーブル (s) に変換するための操作である．

3.2 アクセスコスト見積

アクセスコストは，データベースに格納されているデータを参照するためのリレーション演算（射影，選択，結合など）でアクセスされるハードディスクのブロック数の和であり，データベースとファイルシステムのカタログ情報，および，アクセスログを用いて計算される．表 1 は，蓄積コストとアクセスコスの計算に必要となるデータベースとファイルシステムのカタログ情報の一覧である．

3.2.1 アクセスログを用いたアクセスコスト計算

本研究では，アクセスログをデータ参照のための SQL 文の集合とする．アクセスログである SQL 文 q のアクセスコストの見積は，次のように行われる．

（1） まず，q をリレーション操作から構成される処理木に変換する [1](注1)．

（2） 次に，それぞれのリレーション演算のサイズ（タプルの数）を計算する [1].

(注1)：ビューの展開も行う．

（3） そして，ファイルシステムとデータベースのカタログ情報に基づいて，次の式を用いて，それぞれのタプル数 T をブロック数 B に変換する．

$$B = ceil(\frac{T}{floor(\frac{b-bh}{h+\Sigma_{i=1}^{i=m} size(a_i)})}) \quad (1)$$

ただし，*ceil* と *floor* は，それぞれ，切り上げと切り捨ての関数である．

（4） 処理木のリレーション演算でアクセスされるブロック数（入力となるテーブルのブロック数を含む）の和を求めて，q のアクセスコスト $cost(q)$ とする．

スキーマ s 上のアクセスコストは，s 上のアクセスログ $Q(s)$ に格納されている SQL 文のアクセスコストの総和となる．つまり，スキーマ s のアクセスコストは，次のように計算される．

$$cost_a(s) = \Sigma_{q_i \in Q(s)} cost(q_i) \quad (2)$$

$cost(q_i)$ は，SQL 文 q_i のアクセスコストである．

データベース D のアクセスコスト $COST_{access}(D)$ は，D 上のアクセスログ $Q(D)$ のアクセスコストの和であり，次のように計算される．

$$COST_a(D) = \Sigma_{q_i \in Q(D)} cost(q_i) \quad (3)$$

3.2.2 仮想アクセスコストの見積

再構造化案で構成されるデータベースは実在するものではないため，再構造化案に対応するデータベースの仮想的なカタログ情報とアクセスログは元のデータベースのカタログ情報およびアクセスログから生成する．このような仮想カタログ情報と仮想ログを利用して計算されたアクセスコストを仮想アクセスコストと呼ぶ．本節では，まず，仮想カタログ情報の見積手法，特に，属性の値の種類（以下，属性値のバリエーションという）について説明する．そして，アクセスログの変換手法について述べる．

a） 属値性のバリエーションの見積

表 1 に示されているように，仮想アクセスコスト計算のためのデータベースのカタログ情報は，テーブルに格納されているタプル数，タプルのヘッダサイズ，属性のサイズと属性値のバリエーションがある．そのうち，再構造化で変わるのは，タプル数と属性値のバリエーションである．タプル数は，後述する蓄積コスト見積で計算されるので，その結果を流用する．

ここでは，スキーマの統合操作に応じた属性値のバリエーションの見積手法を述べる．

- 結合 $s = r_1 \oplus r_2$ の場合：属性 a のバリエーションは，次のように計算される．

$$V(s,a) = \begin{cases} V(r_1,a) + V(r_2,a) - \frac{V(r_1,a) * V(r_2,a)}{max(V(r_1,a), V(r_2,a))}, & a \in r_1 \cap r_2 \\ V(r_1,a), & a \in r_1 \\ V(r_2,a), & a \in r_2 \end{cases} \quad (4)$$

- 分割 $s_1 \oplus s_2 = r$ の場合：属性 a のバリエーションは，元と同じとする．つまり，$a \in s_1$ であれば，$V(s_1,a) = V(r,a)$ である．$a \in s_2$ であれば，$V(s_2,a) = V(r,a)$ である．

- 種類変換 $s \sim r$ の場合：属性 a のバリエーションは変わらない．つまり，$V(s,a) = V(r,a)$.

- 新規作成 $s = q(R)$ の場合：本研究では，タプル数の変化に比例して属性値のバリエーションが変化するという考えに基づいて，属性値のバリエーションを求める．つまり，アクセスコスト計算と同じように，SQL 文 $q(R)$ を処理木に変換して，タプル数の変化率を計算して，それを属性値のバリエーションの変化率とする．この手法は，$q(R)$ を実行することなくタプル数を求められるので，DBMS へのアクセスを抑えることが可能である．以下，その手順を示す．

（1） $q(R)$ をリレーション演算から構成される処理木に変換する．

（2） 処理木の葉節点で入力されるタプル数 $T(R)$ と，ルート節点で出力されるタプル数 $T(s)$ を計算する．

（3） 新規スキーマ s の属性 a のバリエーションを次にように計算する．

$$V(s,a) = V(R,a) * \frac{T(s)}{T(R)} \quad (5)$$

$$V(R,a) = max(V(r_1,a), V(r_2,a), ..., V(r_n,a)) \quad (6)$$

ただし，$R = \{r_1, r_2, ..., r_n\}$.

b） アクセスログの変換

データベースの再構造化では，既存のスキーマ集合 $R = \{r_1, r_2, .., r_n\}$ が，スキーマの統合操作によって，新しいスキーマ s に変換される．s 上の仮想アクセスログ $Q(s)$ の生成は，次のように行われる．

（1） R 上のアクセスログ $Q(R)$ から，任意の $r_i \in R$ を含むログを収集して，変換対象集合 $Q_c(R)$ を生成する．

（2） 任意の $q \in Q_c(R)$ を，s 上の質問 q' に変換して，s に対する仮想アクセスログ $Q(s)$ を生成する．

R を s に変換するスキーマ統合操作の種類に応じて，q は次のように q' に書替えられる．

- 結合 $s = r_1 \oplus r_2$ の場合：

（1） q に書かれているスキーマ名"r_1"と"r_2"を新しいスキーマ名"s"に書き換えて，q' を生成する．

（2） q' を整形する．スキーマ名の重複や自明である条件文を省く．

- 分割 $s_1 \oplus s_2 = r$ の場合：

（1） q の *select* 文，*from* 文と *where* 文にあるスキーマ名を次のように書き換えて，q' の *select* 文，*from* 文と，*where* 文をそれぞれ生成する．

（a） 文に書かれている r の属性 A を抽出．

（b） 次のルールに従って，スキーマ名を書き換える．

－ $A \subseteq s_1.A$ であれば，"r"を"s_1"に書き換える．ただし，$s_1.A$ は，s_1 の属性集合を表す．

－ $A \not\subseteq s_1.A$，$A \subseteq s_2.A$ であれば，"r"を"s_2"に書き換える．ただし，$s_2.A$ は，s_2 の属性集合を表す．

－ $A \not\subseteq s_1.A$ かつ $A \not\subseteq s_2.A$ であれば，A を $A_1(\subseteq s_1.A)$ と $A_2(\subseteq s_2.A)$ に分割する．

＊ *select* 文： 属性 $a \in A_1$ であれば，"a"を"$s_1.a$"に書き替える．$a \in A_2$ であれば，"a"を"$s_2.a$"に書き換える．a が A_1

と A_2 の両方に属する場合は，"a"を"$s_1.a$"に書き換える．

＊ $from$ 文："r"を"s_1, s_2"に書き換える．

＊ $where$ 文：属性 $a \in A_1$ であれば，"a"を"$s_1.a$"に書き替える．$a \in A_2$ であれば，"a"を"$s_2.a$"に書き換える．$a \in A_1 \cap A_2$ であれば，"$s_1.a = s_2.a$"を書き加える．

（2） q' を整形する．スキーマ名の重複や自明である条件文を省く．

• 作成 $s = q(R)$ の場合：削除の場合は，関連アクセスログを削除すればよい．

新規作成されるスキーマ s はビューである場合，アクセスコストは変化しないと見なして，アクセスログの変換を行わず，$Q_c(R)$ を $Q(s)$ とする．

s はテーブルであれば，定義の SQL 文 $q(R)$ とアクセスログ q を比較して，q に存在する $q(R)$ に相当する部分を s に書き替えて q' を生成する．具体的に，次のようなステップで変換を行う．

（1） $q(R)$ と q を分割統治戦略 [2] に基づいてそれぞれ処理木 $G(q(R))$ と $G(q)$ へ変換する．

（2） $G(q(R))$ に $G(q)$ が含まれてあれば，その部分を s に置き換えて，$G(q(R))$ の縮小版 $G(q(R,s))$ を得る．

（3） $G(q(R,s))$ を SQL 文に書き直して q' を生成する [1] [2].

• 種類変換 $S \sim R$ の場合：ビューからテーブルへ変換する場合では，q にあるビュー"r"をテーブル"s"に書き替えすればよい．

3.3 蓄積コスト

蓄積コストは，テーブルのデータを蓄積するのに必要となるハードディスクのブロックの数であり，テーブルとファイルシステムのカタログ情報（表 1）に基づいて計算する．

3.3.1 蓄積コストの見積

テーブル $t = (a_1, a_2, \cdots, a_m)$ の必要ブロック数，いわば，蓄積コスト $cost_s(t)$ は，次のように計算される [1].

$$cost_s(t) = ceil\left(\frac{k}{floor\left(\frac{b-bh}{h+\Sigma_{i=1}^{i=m} size(a_i)}\right)}\right) \qquad (7)$$

ただし，t に k タプルがあるとする．

データベース $D = \{s_1, s_2, \cdots, s_n\}$ の蓄積コスト $COST_s(D)$ は，D にあるスキーマ（ビュー以外）の蓄積コストの和である．

$$COST_s(D) = \Sigma_{i=1}^{i=n} cost_s(s_i) \qquad (8)$$

ただし，ビューの蓄積コストは 0 とする．

3.3.2 仮想蓄積コストの見積

既存構成の場合，必要の情報が揃っているため，蓄積コストは式（7）で計算される．しかしながら，新しい構成の場合，データが実在しないため，カタログ情報の見積が必要となる．なお，ここでは，新しいスキーマ（テーブル）の蓄積コストを仮想蓄積コストと呼ぶ．

データベースの再構造化はスキーマ操作によって実現されるため，ファイルシステムへの影響はない．そのため，ブロックサイズと，ブロックのヘッダサイズは変わらない．また，変更後のスキーマの属性のサイズと，タプルのヘッダサイズは既存のスキーマから継承して利用可能である．よって，変更後スキーマのタプルの数を推定すれば，新規スキーマの蓄積コストは計算可能である．

以下，スキーマ操作の種類に応じて，新規スキーマのタプル数の計算式を示す．また，$T(x)$ をスキーマ x のタプル数とする．

• 結合 $s = r_1 \oplus r_2$ の場合：

$$T(s) = T(r_1) + T(r_2) + max(T(r_1), T(r_2))/2 \qquad (9)$$

• 分割 $s_1 \oplus s_2 = r$ の場合：

$$T(s_1) = T(s_2) = T(r) \qquad (10)$$

• 作成 $s = q(R)$ の場合：削除の場合の蓄積コストの計算は行わない．作成されるスキーマ s はビューである場合，蓄積コストを 0 とする．

s はテーブルであれば，s を定義する SQL 文で返される結果のタプル数を新規テーブルのタプル数とする．

• 種類変換 $s \sim r$ の場合：ビューからテーブルへの変換であるため，テーブルの新規作成と同じ手法で計算できる．すなわち，ビューの定義の SQL 文を解析して，その SQL 文で返されるタプル数を変更後スキーマのタプル数とする．

3.4 統合コストの見積

スキーマ s の統合コスト $COST(s)$ は，次のように計算される．

$$cost_{int}(s) = \alpha * cost_s(s) + \beta * cost_a(s) \qquad (11)$$

ただし，α と β は，ユーザ調整可能の重み付けパラメータである．

同様に，データベース D の統合コスト $COST_{int}(D)$ は，次のように計算される．

$$COST_{int}(D) = \alpha * COST_s(D) + \beta * COST_a(D) \qquad (12)$$

4. 可視化手法

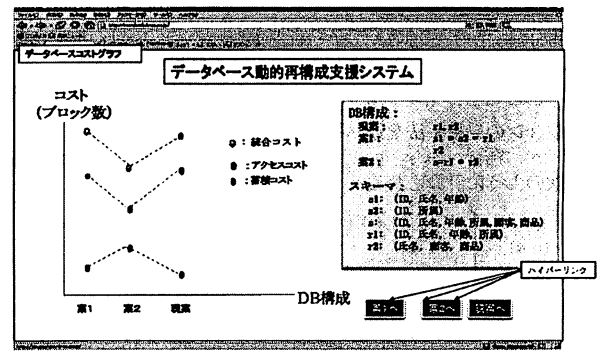

図 2 データベース構成のコスト比較ページの表示例

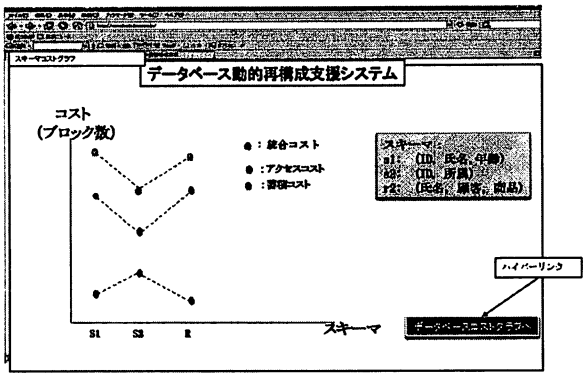

図 3　スキーマのコスト比較ページの表示例

2. 節で説明したように，見積コストの可視化は，グラフ化と Web ページ生成の二つのステップに分けられている．つまり，コスト見積の結果を表示するためのグラフと，それを表示するための Web ページをそれぞれ生成する．

データベースのコストグラフの縦軸は，ブロック数を単位とするコストであり，横軸はデータベースの構成案を示している．一方，スキーマのコストグラフは，縦軸は同じくコスト（ブロック数）であるが，横軸はスキーマである．コストグラフに，蓄積コスト，アクセスコストと統合コストがそれぞれプロットされる．

データベースとスキーマのコストグラフ，および，これに対応するデータベース構成情報とスキーマ情報を含む Web ページ群が生成される．これらのページは，データベースの構成とスキーマの対応関係に基づいて生成されるハイパーリンクで関連づけられる．図 2 と図 3 は，それぞれ，データベース構成のコスト比較ページとスキーマのコスト比較ページの表示例である．

5. 実　　験

提案手法の検証のため，我々は，3つの実験を行った．実験 1 では，データベースの再構造化効果の確認を，実験 2 および実験 3 では，提案した蓄積コストとアクセスコストの見積手法の評価をそれぞれ行った．

5.1 実験環境

表 2　テストデータ（DB とアクセスログ）の構成

セット	利用 DB	ログ (ベース+追加)	対応統合操作
A	DB1	230 文+56 文	結合：$s_1 = customer \oplus address$
B	DB2	1360 文+120 文	分割：$s_2 \oplus s_3 = customer$
C	DB2	1360 文+40 文	新規作成:$s_4 = q$
D	DB2	1360 文+40 文	種類変換 $s_5 \sim r$

実験マシンの基本スペックは，CPU Xeon 3.2G，メモリ 4GB, HDD 640GB（RAID5），CentOS 4.4 であった．実験では，DBMS として MySQL を利用し，サンプルのデータとして OSDL の DBT1 と DBT2 から作成した以下の 2 種類のものを利用した．

- DBT1 のテーブル country，shopping_cart と shopping_cart_line を省いて，7 つのテーブルから構成されたデータベース DB1
- DBT2 の 9 つのテーブルを流用したデータベース DB2

また，DBT1 と DBT2 のデータを利用して，それぞれ，230 文と 1360 文のアクセスログ（ベースログ）を生成した．また，スキーマ統合操作の種類に応じて，さらにアクセスログ（追加ログ）を追加して，4 種類のアクセスログを作成した．これらの追加ログによって，データベースの再構成案が作成されるという想定である．実験で用いた DB とそのアクセスログの構成は，表 2 に示されている(注2)．

5.2 実　験　1

表 3　再構造化の効果確認実験の結果

セット	アクセス時間 (秒)		蓄積容量 (バイト)	
	再構築前	再構築後	再構築前	再構築後
A	53515.12	2836.75	3103424712	4320157500
B	22.19	26.55	80711976	80525260
C	9.97	7.27	80711976	82959300
D	10.02	7.19	80711976	82959300

実験結果を表 3 に示す．これは，変更される前後のデータベースのパフォーマンス（アクセス時間と蓄積容量）の比較を行ったもので，再構成のためのスキーマ統合操作は，表 2 で示されている 4 種類である．アクセス時間は，すべての質問（ログ）の処理時間の和で，アクセスコストは，全てのデータファイルのサイズの和である．また，スキーマ統合操作に応じて，実際の DB を再構築した．新しい構成におけるアクセスログは，元の質問を元に手動で作成している．これらの結果から，再構造化による蓄積容量とアクセス時間の増減効果は一定でないことが分かる．例えば，テーブルを分割してアクセス時間を早くすることがよくあるが，テストセット B で示されているように，場合によってアクセス時間が増加することもありうる．これにより，見積の重要性が分かる．

5.3 実　験　2

表 5　蓄積コストの増減率の比較

	A	B	C	D
見積った増減率	0.379	-0.002	0.0279	0.0279
実際の増減率	0.392	-0.0007	0.0283	0.0283

実験 2 では，テーブル毎に，実際のデータファイルのサイズと見積ったサイズを比較し，見積結果の誤差を求めた．見積結果の誤差率は，次の式で計算される．

(注2)：$s_2 = \{c\_id, c\_d\_id, c\_w\_id, c\_first, c\_middle, c\_last\}$, $s_3 = \{c\_id, c\_street\_1, c\_street\_2, c\_city, c\_state, c\_zip, c\_phone, c\_since, c\_credit, c\_credite\_lim, c\_discount, c\_balance, c\_ytd\_payment, c\_delivery, c\_date\}$. q は，"create table s as select c_id, c_d_id, c_w_id, c_first, c_middle, c_last, c_street_1, c_street_2, c_city, c_state, c_zip, c_phone from customer" である．r はテーブル customer から生成されたビューであり，"create view r as select c_id, c_d_id, c_w_id, c_first, c_middle, c_last, c_street_1, c_street_2, c_city, c_state, c_zip, c_phone from customer"．s_4 は r から変換されるテーブルである

表 4　蓄積コスト見積の実験結果

DB/セット	実際のブロック数	データタイプによる属性サイズ推定				データの平均長による属性サイズ推定			
		floor 関数有効		floor 関数無効		floor 関数有効		floor 関数無効	
		見積結果	誤差率	見積結果	誤差率	見積結果	誤差率	見積結果	誤差率
dbt1	757677	49860151	64.81	49191979	63.92	785386	0.04	777340	0.03
dbt2	19711	527630	25.77	499118	24.32	21384	0.08	20231	0.03
A	1054310	98589751	92.51	93436922	87.62	1123671	0.066	1072237	0.02
B	19666	528001	25.84841859	499119	24.37979254	21041	0.069917624	20216	0.03
C	20260	529505	25.14	500980	23.73	21961	0.08	20803	0.03
D	20260	529505	25.14	500980	23.73	21961	0.08	20803	0.03

$$\text{誤差率} = \frac{(\text{見積ブロック数} - \text{実際ブロック数})}{\text{実際ブロック数}} \quad (13)$$

実験では，属性のサイズをデータタイプの規定サイズで計算する場合と，実際のデータの平均長さで計算する場合に分けている．タプルとブロックのヘッダサイズは，共に 4B にした．ブロックのサイズは 4KB にした．また，式 (7) の切り下げ関数 *floor* を有効と無効に切り替えながら実験を行った．実験結果を表 4 に示す．なお，データファイルのブロックサイズは，"ls" コマンドより取得されたものである．

実験結果から，実際のデータの長さから推定した属性サイズを用いてタプルサイズを計算する方が，誤差率が低いことが分かる．これは，データタイプによる属性サイズを推定する時に，そのデータタイプの許される最大値を用いたからである．例えば，MySQL などでは，text 型のサイズが最大 6KB と規定されているが，実際のデータでは，その 1 割以下の場合が多い．そのため，6KB でタプルサイズを計算し，ブロック数を多く見積ってしまう可能性が非常に高い．また，floor 関数無効の場合のほうが有効の場合より誤差率が低いことも分かる．ファイルシステムのブロック長が動的に決定されていることが原因であると思われる．実際のファイルシステムと DBMS のストレージエンジンの特徴を考慮して蓄積容量を見積ることが重要であることが分かる．

また，再構造化前後の蓄積容量の増減率を次のように求めて，その結果を表 5 に示している．

$$\text{増減率} = \frac{(\text{見積コスト} - \text{実際コスト})}{\text{実際コスト}} \quad (14)$$

この結果から，再構成によるデータ容量の増加または減少が正しく見積られたことが分かる．また，その増減率の差も 2%以下であり，一定精度の見積ができたと思われる．

5.4 実　験　3

表 6　アクセスコストの増減率の比較

	A	B	C	D
見積った増減率	-0.998	0.906	-0.016	-0.016
実際の増減率	-0.946	0.197	-0.271	-0.284

実験 3 では，我々は，テストデータセット A，B，C と D に対して，再構造化前後のデータベースにおける質問の実行時間の増減率と，見積ったアクセスコストの増減率を計算して比較を行った．増減率は，式 (14) で計算される．また，実験 2 の結果から，実際のデータの長さでタプルのサイズを計算している．なお，質問の実行時間は，MySQL++を利用して開発したツールで測った．表 6 ではその増減率を示す．アクセス時間の増減傾向が正しく見積られているが，結合以外では，見積と実際の増減率の差が非常に大きく，精度の改善が必要であると思われる．今後，これらの実験結果のさらなる分析を行うと共に，データサイズ，索引およびジョイン順序などタプル数以外要素も考慮したアクセスコストの見積手法について検討する．質問変換による仮想ログの生成についてもさらに検討する．例えば，既存のテーブルの一部を用いて新しいテーブルを生成した場合，問合せ先として既存のテーブルと新しいテーブルのどちらも可である場合はあるが，適切なテーブルの選択手法が必要である．

6. おわりに

本稿では，データベースの再構造化を支援するためのコストの見積と可視化手法を提案し，これらの手法を利用したプロトタイプシステムを紹介した．実験結果から，蓄積コストの見積が，実際の容量の増減率との差が 2%以下に抑えられることがわかった．一方，アクセスコストの見積手法は，アクセス時間の増減傾向を正しく予測できたが，実際のアクセス時間の増減率との差が大きく，改善する必要がある．

今後，実験結果のさらなる分析と提案手法の改良を行う予定である．また，提案手法を拡張して，実際のデータ利用形態，リソース制約とデータの特性に応じたデータベースの最適構成の自動選別手法についても開発する．

文　献

[1] Hector Garcia-Molina, Jeffrey D.Ullman, Jennifer Widom 著, Database Systems: the Complete Book, Prentice Hall, New Jersey, 2002

[2] C.J.Date 著，藤原譲監訳：データベースシステム概論 原書 6 版, 丸善株式会社，東京，1997

[3] Daniel C. Zilio, Calisto Zuzarte, Sam Lightstone, Wenbin Ma, Guy M. Lohman, Roberta Cochrane, Hamid Pirahesh, Latha S. Colby, Jarek Gryz, Eric Alton, Dongming Liang, Gary Valentin: Recommending Materialized Views and Indexes with IBM DB2 Design Advisor. Proc. of ICAC 2004, pp.180-188, 2004.

[4] Harumi A. Kuno, Elke A. Rundensteiner: Materialized Object-Oriented Views in MultiView. Proc. of RIDE-DOM 1995, pp. 78-85, 1995

[5] Josep Silva, Jorge Belenguer, Matilde Celma: Multi-source Materialized Views Maintenance: Multi-level Views, Proc. of ADBIS 2006, pp. 71-80, 2006

[6] David DeHaan, Per-Ake Larson, Jingren Zhou: Stacked indexed views in microsoft SQL server. proc. of SIGMOD Conference 2005, pp. 179-190, 2005.