DEWS2007 E7-8

管理者に対しても機密を保持できる暗号化データベースの索引構成法

三浦　志保†　　渡辺知恵美††

† お茶の水女子大学理学部情報科学科　〒 112-8610 東京都文京区大塚 2-1-1
E-mail: †shiho@db.is.ocha.ac.jp, ††chiemi@is.ocha.ac.jp

あらまし　近年，データベース製品の管理運用を外部のデータベース技術者に委託するデータベースアウトソーシングサービスが普及しつつある．このときサービスの利用者は，機密事項であるデータを外部のデータベースサービスプロバイダに委託しなければならない．利用者はデータの内容をデータベース管理者に閲覧されたくないという要求を持つ．そのため，管理者に対しても機密を保持できる暗号化データベースシステムの研究が現在進められている [4]．このシステムでは，クライアント側でタップルごとにデータを暗号化し，それに索引を付与してサーバに置く．索引には，ドメイン領域を分割してそれぞれに割り当てられたバケット番号が格納される．検索時は以下の 2 段階問合せが行われる．まずフィルタリング処理として，タップルに付与された索引を用いてサーバで大まかな検索を行う．そして，該当したタップルに対してのみクライアント側で解精製を行う．文献 [4] では，バケットのデータ分布から索引情報が推測される可能性がある．よって本稿では，MaxDiff ヒストグラムに基づいた，より安全な索引構成法を提案する．
キーワード　データベースセキュリティ，プライバシー保護，暗号，ヒストグラム

A Indexing Method for Encrypted Databases Protecting Confidential Data even from DBAs

Shiho MIURA† and Chiemi WATANABE††

† Department of Information Sciences, Faculty of Science, Ocyanomizu University
E-mail: †shiho@db.is.ocha.ac.jp, ††chiemi@is.ocha.ac.jp

Abstract　Recently, the database out-sourcing service model which entrusts database management to database service provider has been gaining much attention.To use this model, user must entrust confidential data to external database administrator. Then users of the system require to protect their data from even database administrator. To resolve the problem, encrypting database system has been proposed [4]. Each tuple is encrypted and added privacy-preserving indexes. The indexes are created from bucket-id which divides their domain area. The query operator processes in the following two steps; (1) filtering query using privacy-preserving indexes in server, and (2) refinement query from decrypted tuples in client. The index method used in paper [4] is about the possibly of deduction from the bucket distribution. We propose a safety-higher bucketizing method based on MaxDiff histogram.
Key words　database-security, privacy-preservation, cryptography, histogram

1. はじめに

近年，個人情報保護法の制定や大規模な情報漏洩事件が次々に明るみに出たことにより，データベースセキュリティに対する意識が急速に高まっている．多くの DBMS 製品は豊富なセキュリティ機能を備えているが，原則としてデータベース管理者に全面的な信頼を置くことを前提に提供されている．そのため，管理不十分による情報漏洩や管理者自身の内部犯行に十分に対処できない．

また，データベース製品の管理運用を外部のデータベース技術者に委託するデータベースアウトソーシングサービスも現在普及しつつある．このときデータサービスの利用者は，委託業者の管理者に機密事項であるデータの内容を閲覧される可能性をなくしたいという要求を持つ．

そこで近年では，暗号化したデータベースと索引をサーバに置くことによって利用者側が管理者に頼ることなく情報漏洩を防ぐことのできる暗号化データベースシステムの研究が進められている [4]．暗号化データベースシステムでは，クライアント側でタップルごとにデータを暗号化し，セキュアな索引を付与し，それらをサーバに置く．検索時には，まずサーバで索引を

用いて大まかな検索を行ってから該当タップルをクライアント側で復号化し，解の精製を行う．このときタップルに付与する索引は，タップルの内容が侵入者に推測されず，かつ，サーバ側でのデータの絞込みが十分に行われるものでなければならない．これまで用いられている索引構成法では，属性のドメイン領域を等間隔のバケットに分割し各バケットにランダムに付けられたバケット番号をその属性の索引値とする方法が用いられている（その詳しい説明は 2 節に述べる.）しかしながらこの手法では属性値に偏りがある場合，各バケットに入るタップル数の関係から値が推測されてしまう可能性があり十分に安全であるとはいいがたい．

そこで本研究では，タップルの内容を侵入者に推測されにくいより安全な索引構成法を提案する．この索引構成法は，暗号化データベース内のテーブルを見ることができ，かつログも見ることができるデータベース管理者に対しても機密を保持できるものでなければならない．本提案手法は MaxDiff を用い，タップルの挿入または変更時に，各バケットに入るタップル数が均等になるようにバケットを調整する．またダミーデータを効果的に用いることにより，ログや索引を分析しても値が推測できないようにする．本稿ではその初期的試みとして単一属性を対象にタップル挿入時のバケット調整法について提案しその検証を行った．まず第 2 節にて暗号化データベースについて説明した後，第 3 節で安全な索引構成法を提案し，バケット調整の素朴法について述べる．素朴法は各バケットに入るタップル数が均等にはなるものの実行時間などの問題がある．それらの問題についての検証を第 4 節で行った後，バケット内ヒストグラムを用いた解決法について述べ，実行時間が十分に短縮されることを検証する．第 5 節は関連研究，第 6 節はまとめと今後の課題である．

2. 暗号化データベースシステム

2.1 暗号化データベースに対する処理の流れ

図 1 にクライアントとサーバにおける，暗号化データベースに対する処理の流れを示す．データは全てクライアント側で暗号化されてから，サーバ側の暗号化データベースに格納される．これで例えデータベース管理者であっても，タップルの内容を知ることはできなくなる（図 1①）．暗号化データに対する検索は，データに付与したセキュアな索引を用いて後に説明する 2 段階問合せを行うことにより実現する（図 1②）．またサーバ側には，暗号化データベースの他に索引情報（図 1 ★）を置き，クライアントがデータを挿入・更新したり検索したりする際には，自動的にここから情報を取得できるようにする．

2.2 暗号化と索引の付与

暗号化データベースのテーブルは，元データのタップルを暗号化した文字列と各属性の索引とで構成されている．リレーション R において，A_iを 属性，A_i^S をその属性に対する索引，etuple を 1 つのタップルを暗号化した文字列とすると，下のように表すことができる．

$\mathrm{R}(A_1, A_2, \cdots, A_n) \Rightarrow R^S(\mathrm{etuple}, A_1^S, A_2^S, \cdots, A_n^S)$

各属性の索引を作成するにあたって，まずは，その属性のドメイン領域を適当に分割する（以下，分割された 1 つ 1 つの領域のことをバケットという.）

$\mathrm{partition}(\mathrm{R}.A_i) = \{p_1, p_2, \cdots, p_k\}$

次にそれぞれのバケット p_j にバケット番号をハッシュ関数により与え，その値 m を索引値として格納する．

$ident_{R.A_i}(p_j) = \mathrm{m}$

また，それぞれのバケットの範囲，割り当てられたバケット番号は索引情報として別に格納する．索引情報もまたクライアント側で暗号化してから，サーバ上におく．

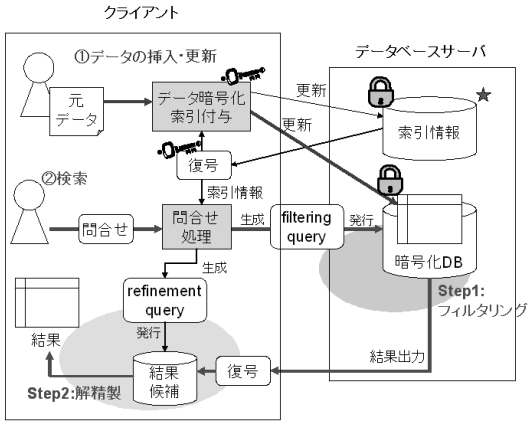


図 1　暗号化データベースに対する処理の流れ

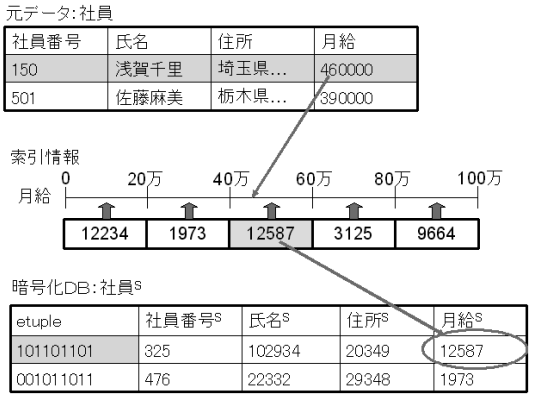


図 2　暗号化と索引の付与

社員というテーブルで例を図 2 に示す．まず，社員 (社員番号, 氏名, 住所, 月給) ⇒ 社員^S($etuple$, 社員番号^S, 氏名^S, 住所^S, 月給^S) とし，社員^S をサーバ側にある暗号化データベースに格納する．1 行目の etuple には “101101101”が格納されているが，これは “150 浅賀千里 埼玉県...460000” を暗号化したものである．索引を作成する方法を月給を例に考えた場合，まず，$\mathrm{partition}(\text{社員.月給}) = \{(0, 20\text{万}], (20\text{万}, 40\text{万}], (40\text{万}, 60\text{万}], (60\text{万}, 80\text{万}], (80\text{万}, 100\text{万}]\}$ と領域を 5 つのバケットに分割する．そして，各バケットに $ident_{\text{社員.月給}}((40, 60\text{万}]) = 12587$ のようにバケット番号を与える．したがって、月給=460000 は 月給^S = 12587 として格納される．

2.3 検　　索

ユーザが発行した問合せ文は，以下の 2 段階の問合せに分割され実行される．

- filtering query：サーバ側の索引に対する問合せ（フィルタリング処理）により，大まかな結果候補を抽出する．

- refinement query：クライアント上で復号化した結果候補（フィルタリング処理により抽出）に対して解精製処理を行い，最終的な結果を求める．

例えばクライアントが，月給が 55 万円以上の社員の名前を求めるために「SELECT 名前 FROM 社員 WHERE 月給 $\geqq$ 550000」という問合せ文を発行したとする．その場合，まずはサーバから索引情報（バケット情報）を自動取得してきて復号化を行い，その情報を元に filtering query「SELECT etuple FROM 社員^S WHERE (月給^S = 12587 or 月給^S = 3125 or 月給^S = 9664)」を生成する．この filtering query をサーバに発行することによって，月給が 55 万円以上の可能性のある社員のタップルを検索する．上で述べた通り，この結果は大まかな結果候補にすぎない．filtering query では「WHERE (月給^S = 12587 or 月給^S = 3125 or 月給^S = 9664)」としているので，厳密には月給が 40 万円以上の社員のタップルが返ってくるからである．本当の結果を求めるために，このフィルタリングされた結果候補をクライアント側で復号化した後に，refinement query 「SELECT 名前 FROM 社員 WHERE 月給 $\geqq$ 550000」を発行する．この作業によって，真の結果を得ることができる．

この 2 段階の問合せの仕組みの大きな特徴は，サーバ側でフィルタリング処理をすることによりクライアント側での処理が少なくてすむため，検索効率をさほど下げることなく安全性を保てるという点である．

3. 安全な索引構成法の提案

3.1 従来の索引構成法による安全性

暗号化タップルに付与する索引は，タップルの内容を侵入者が推測できないものでなければならない．しかし，文献 [4] で用いられている等間隔の領域分割法では，元データにおいて同じ値（または近い値）同士のものは同じバケットに入るため，暗号化データベース内でも同じ値で格納される．さらに，それぞれのバケットに入っているタップル数は一目で分かる．したがって，それぞれのバケットに入っているタップル数に偏りがある場合，分布解析などによりデータの内容を予測されてしまう可能性がある．例えば給与の場合，高額の給与をもらっている人はそれほど多くないので，各バケットに入っているタップル数には偏りが生じる．

サーバ側にたくさんのサンプルが与えられたとしても，予測できない索引構成方法を考える必要がある．そこで我々は MaxDiff [7] による領域分割をベースにした索引構成法を提案する．

3.2 索引構成法の提案

3.2.1 基本概念

本研究では，MaxDiff [7] での領域分割法を用いる．これは，1 つのバケットに入るタップル数の差が閾値以下になるように分割する方法である．この方法を用いることにより，バケットによるタップル数の偏りが生じなくなるため，分布解析からデータの内容を予測されるのを防ぐことができる．

具体的には，データが挿入される度にそれぞれのバケットの範囲を変えることにより，それぞれのバケットに入るタップル数に一定数以上の差を生じさせないようにする．以下，それぞれのバケットに入るタップル数の差の上限を MaxDiff と表す．バケットを分割する際，バケットの個数の上限を決めておく方法と，バケットに入るタップル数の上限を決めておく方法が考えられるが，本稿ではバケット分割にかかるコストを考えバケットの個数の上限を決めておく方法について述べる．以下，バケットの個数の上限を p_{max} と表す．各バケットに入るタップル数の上限を決めた分割法の提案や，本提案法との比較は今後の課題として行う予定である．

3.2.2 データ挿入時のバケット調整法

データ更新の際に使う，索引情報テーブルの例を表 1 に示す．ただし，表 1 は図 2 に示したテーブル ‘社員’ の索引情報から，属性 ‘月給’ の部分のみを示している．下限値，上限値とはあるバケットにおける上限と下限を表しており，バケット番号とはそのバケットに割り当てられている値である．また，表 1 に示しているのはクライアント側で復号化した後の状態であり，実際にサーバ側に格納しているときは，下限値・上限値・バケット番号・タップル数はまとめて暗号化した文字列となっている．

テーブル名	属性	下限値	上限値	バケット番号	タップル数
社員	月給	0	20 万	12234	19
社員	月給	20 万	40 万	1973	20
社員	月給	40 万	60 万	12587	21
社員	月給	60 万	80 万	3125	21
社員	月給	80 万	100 万	9664	19

表 1　索引情報テーブル

クライアントがデータ挿入したときの処理の流れを以下に示す．

(1)　サーバから索引情報を取得し復号化する．
(2)　データ挿入後，必要ならばバケット調整を行い，索引情報および暗号化データベースを更新する．

上記の (2) における，具体的な更新方法について説明する．

タップル r をリレーション R に挿入する際，リレーション R の各属性 $R.A_i(1 \leqq i \leqq n)$ に対して，partition($R.A_i$)=($p_0, p_1, \cdots, p_k$) の領域を以下の手法に従って調整し，属性 $R.A_i$ の値がバケット $p_j(1 \leqq j \leqq k)$ の範囲に含まれるタップルの数 num(p_j) の差が一定数以下となるようにする．以降 $R.A_i$ の領域調整について述べるため，「属性 $R.A_i$の値がバケット p_jの範囲に含まれるタップル」を，単に「p_j の該当タップル」と呼ぶことにする．各バケット p_j は (下限値, 上限値] と表し，まだデータが入っていない状態でも，初期値として，p_0 = ($R.A_i$のドメインの下限値, $R.A_i$のドメインの上限値] が与えられる．$bottom(p_j)$ はバケット p_jの下限値, $top(p_j)$ はバケット p_jの上限値を示すものとする．

今，MaxDiff=2，p_{max}=5 として，“insert into 社員 values (‘200’ , ‘柳平有美’ , ‘東京都. . . ’ , ‘610000’)” が発行されたときの更新方法を属性 ‘給与’ を例に示す．

(1)　バケットの個数 $< p_{max}$ のとき

分割数がまだバケット数の上限に達してないときは，基本

的に 1 バケットに 1 タップルとなるようにデータ挿入のたびにバケットを分割する．上記の例では属性値 ‘610000’ が該当するバケットを探し，図 3 のように，そのバケット p_3 を (bottom(p_3),610000] と (610000,top(p_3)] に分割する．2 分割したうち小さい方のバケットに ‘610000’ を格納する．結果各バケットの上限値のところにのみデータが存在する状態となる．また挿入する値がバケットの上限値である場合は分割ができないためそのまま挿入する．

挿入後，バケット内タップル数の最大値とバケット内タップル数の最小値との差が MaxDiff を超える場合（例では，各バケットの該当タップル数に 3 以上の差が出てしまった場合）がある．そのときは該当タップル数が最小であるバケットにダミータップルを挿入することで調整を行う．

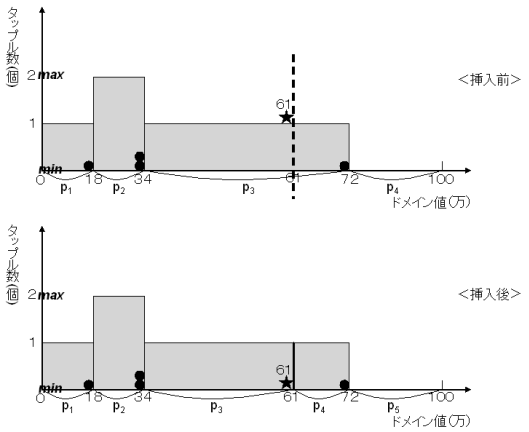


図 3　バケットの 2 分割

(2)　バケットの個数 $= p_{max}$ のとき

バケット数がその上限 p_{max} を満たしている場合も，まずは挿入する値 ‘610000’ が該当するバケットを探し，そのバケットに格納する．格納後，バケット内タップル数の最大値と最小値との差が MaxDiff を超えない場合は，バケット調整は行わない．

タップルの挿入後に，バケット内タップル数の最大値と最小値との差が MaxDiff を超えてしまう場合，該当タップル数が最大であるバケット p_j に対しバケットの切り崩しを行う．

[バケットの切り崩し]

バケット p_j の隣接バケットを見て，該当タップル数の少ない方のバケット p_l（$p_l = p_{j-1}$または p_{j+1} とする）へ p_j 内のタップルを切り崩す．該当タップルの切り崩しは p_jおよび p_l の範囲を変えることによって行う．$p_l = p_{j+1}$ である場合はバケット p_j をスキャンし，p_j 内で 2 番目に大きな値を p_j の上限値および p_{j+1} の下限値とすることによって p_j 内の最大値を p_{j+1} に移動する．$p_l = p_{j-1}$ である場合はバケット p_j 内の最小値を求め，p_{j-1} の上限値および p_j の下限値とすることによって p_j 内の最小値をバケット p_{j-1} に移動する．

ただし，p_jの範囲を変更することにより, p_jの該当タップル数が最小になってしまう場合は その処理を行わず，逆隣への切り崩しを同様に試みる（これを以下，方向転換と呼ぶ）

p_jの切り崩しにより，隣接バケット p_lの該当タップル数が最大となる場合は, p_lに対して切り崩し処理を行う．

この切り崩し処理を，各バケットの該当タップル数の差が MaxDiff 以下となるか，切り崩し処理がそれ以上できなくなるまで繰り返す．

図 4 ではバケット p_3 からバケット p_2 への切り崩しをするために，p_3 内の最小値である 53 を p_2 の上限値および p_3 の下限値とする．しかし，この処理により p_2 の該当タップル数が 4 となり，p_5 の該当タップル数との差が MaxDiff=2 を超えるため，p_2 に対して切り崩し処理を行い，p_1, p_2 の範囲を変えることによって，p_2 の該当タップルの内 1 つを p_1 に渡している．

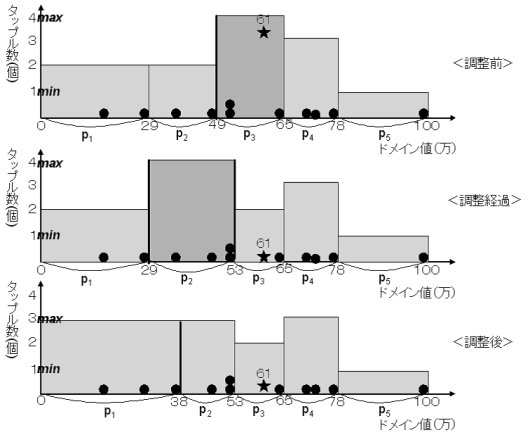


図 4　バケットの切り崩し

図 5 は，方向転換が必要な例である．まずバケット p_4 からバケット p_3 への切り崩しを試みるが，p_4 内の最小値を持つタップル数は 3 個あり，これらを p_3 に渡した場合，p_4 の該当タップル数は明らかに最小になってしまう．したがって p_4 から p_3 への切り崩しをせずに方向転換をし，バケット p_5 の方へ p_4 内の最大値を持つタップルを渡している．

該当タップル数最大のバケットの切り崩しを上記の通りに行った後も，その差が MaxDiff 以下にならない場合が考えられる．その場合，該当タップル数が最小である領域 p_j を見つけ，バケットの埋め合わせを行う．

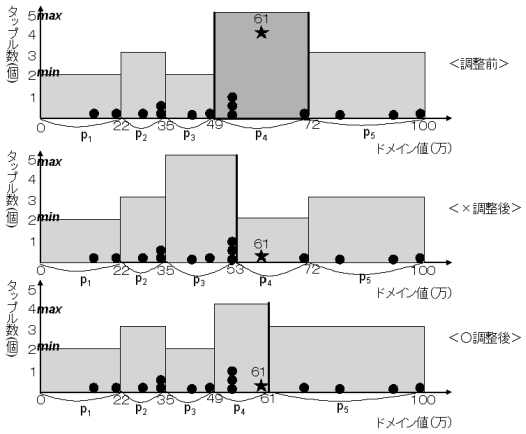


図 5　方向転換を含むバケットの切り崩し

[バケットの埋め合わせ]

該当タップル数最小のバケット p_j の隣接バケットのうち該当

タップル数の多い方からバケット p_j へタップルを切り崩すことにより，p_j の該当タップル数を増やす．ただし，切り崩しの時と同様、タップルの受け渡しにより隣接バケットの該当タップル数が最小となる場合は方向転換を行う．

[ダミータップルの挿入]

バケットの切り崩しと埋め合わせを行っても各バケットの該当タップル数の差が MaxDiff を超える場合は該当タップル数が最小であるバケットにダミータップルを挿入することで調整をはかる．

本システムでは，ダミータップルが増えないための工夫が施されている．１つ目は，ダミータップルの上書きである．すでにダミータップルが格納されているバケットに新しい値が格納された場合，そのダミータップルを上書きする．そうすることで無駄にデータ量を大きくすることを防いでいる．２つ目は，ダミータップル数をそのバケットの該当タップル数として数えないことである．バケット内タップル数の最大値と最小値との差が MaxDiff を超える場合，切り崩し処理や埋め合わせ処理によりバケット調整を行うが，始めに該当タップル数を数える時点で，ダミータップルを数えないことで無駄なバケット調整を行うのを防ぐことが出来る．よってダミータップルが必要以上に増えるのを防ぐことが出来る．

4. 提案手法の改良

4.1 提案手法における懸案事項

前章で提案した索引構成法には大きく分けて４つの問題点が考えられる．

(1) バケット p_j の該当タップル数を調整する場合，p_j 内の最小値もしくは 2 番目に大きな値を求めるために，p_j 内のタップルデータを全てサーバ上の暗号化テーブルから取得し復号化しなければならない．よって，サーバから取得するタップル数が膨大だった場合，実行時間も膨大となる可能性がある．

(2) １個のデータを挿入する度に，バケット調整においてバケット番号が変わった複数のタップルを更新しなければならない．更新タップル数が膨大になった場合，実行時間も膨大となる可能性がある．

(3) 本来，各バケットに与えられるバケット番号はランダムなため，サーバ側ではバケットの隣接関係は分からないようになっている．しかしデータ挿入に伴うタップルの更新パターンによって，バケットの隣接関係が推測できる可能性がある．

(4) 各バケットにおける該当タップル数調整のために，ダミーデータが増えて必要以上にデータ量が大きくなる可能性がある．

4.2 検　　証

前節で挙げた４つの問題点の影響を調べるため，それぞれ４つの実験を行った．４つの実験において挿入したデータは，いずれも一様乱数により生成した値である．また，属性のドメインは [0,50000]，最大バケット数 p_{max} は 20，maxdiff は 2 である．

実験 (1)　サーバから取得するタップル数と時間の連動性

1000 回のデータ挿入を行い，各挿入処理時バケット調整のためにサーバから取得したタップル数と実行時間を計測する．

実験 (2)　更新タップル数と実行時間の連動性

1000 回のデータ挿入を行い，データ挿入に伴うバケット調整による更新したタップル数と実行時間を計測する．

実験 (3)　タップルの更新パターン

1000 回のデータ挿入を行い，データ挿入に伴う各バケットの更新タップル数を調べる．

実験 (4)　ダミーデータ

100 回のデータ挿入を行い，各バケットのダミータップル合計数を調べる．これを 10 回繰り返し，ダミータップル合計数の平均を求める．

図 6 に実験 (1) の結果を示す．図 6(a) が実行時間，図 6(b) がサーバから取得したタップル数のグラフである．グラフからわかるように，サーバから取得したタップル数が多いとき，実行時間も大きくなっている．これは暗号化テーブルを全てスキャンし，該当するタップルを復号化するのにコストがかかっているからだと考えられる．

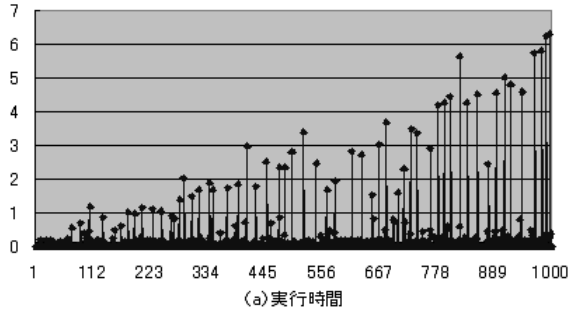
図 6(c) に実験 (2) の結果 (更新タップル数) を示す．更新タップル数の増え方と実行時間の増え方にはずれが生じている．このことから，実行時間が更新タップル数よりサーバから取得するタップル数に依存していることが分かる．サーバから取得しなければならないタップル数は普通，更新タップル数よりも大幅に多い上に，復号化作業が伴うからである．

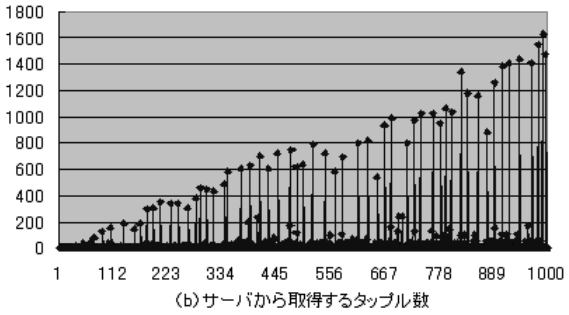
図 7 に実験 (3) の結果の一部を示す．1000 回のデータ挿入のうち 37 回目から 46 回目のデータ挿入の結果を示しており，またバケットも全 20 個のうち p_2 から p_{12} までの 11 個のみ示している．ただし，示されていないバケットに関してはタップルの更新がなかったものとする．図から 37，38，39，41，43 回目の挿入では，タップルの更新がないことが分かる．これはデータの挿入後，バケット調整を行う必要がなかった場合だと考えられる．また 40，42，44 回目の挿入では，データを挿入したバケットの隣のバケットのタップルが更新されている．これは挿入後，１回バケット調整が行われた場合だと考えられる．また 45，46 回目の挿入では，データを挿入したバケットの隣のバケットから遠いバケットまで連続してタップルが更新されている．これは挿入後、複数回バケット調整が行われた場合だと考えられる．

全体（1000 回）を通して最も多いのは，データを挿入したバケットの隣のバケットのタップルが更新されるパターンである．またサーバにはどのバケットにデータが挿入されたかがログとして残ってしまう．そのためタップルが更新されたバケットと，ログから分かるデータを挿入したバケットとが隣接関係にある可能性が高いと予測することができてしまう．

図 8 に実験 (4) の結果を示す．図 8(a) は 10 回分の施行におけるダミータップル数の合計値である．trial3，6，30 以外はほとんどダミータップルがなく，均等に分けられていることがわかる．しかし trial3，6，30 においてはダミー数が非常に多くなった．最もダミータップルの多い trial3 における各バケットのダミータップル数を図 8(b) に示す．この場合は p_1 から p_7 のタップル数に合わせる形で p_8 以降のバケットにダミータップルを含めている．本手法ではタップルを挿入するたびに差分的

にバケット調整を行っているが，挿入する値に偏りがある場合に切り崩しがうまくいかずダミータップルが増える結果になると思われる．また，バケット数 p_{max} を小さく設定したり，許容するバケットごとの該当タップル数の差 MaxDiff を小さく設定したりした場合は，バケット調整による高さの均等化が難しくなり，ダミータップル数が大幅に増える可能性がある．また，分布的に偏りのあるデータを挿入していった場合も同様にしてダミータップル数が増える可能性がある．





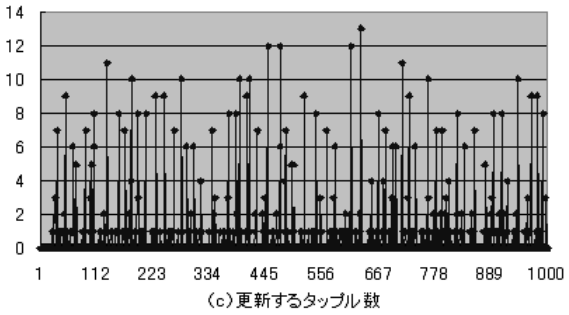


図 6　素朴法の実験結果（(a) 実行時間，(b) サーバから取得するタップル数，(c) 更新するタップル数

挿入回数	挿入バケット	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
37回目	p_9	0	0	0	0	0	0	0	0	0	0	0
38回目	p_6	0	0	0	0	0	0	0	0	0	0	0
39回目	p_{10}	0	0	0	0	0	0	0	0	0	0	0
40回目	p_{10}	0	0	0	0	0	0	0	0	0	1	0
41回目	p_{12}	0	0	0	0	0	0	0	0	0	0	0
42回目	p_6	0	0	0	1	0	0	0	0	0	0	0
43回目	p_7	0	0	0	0	0	0	0	0	0	0	0
44回目	p_7	0	0	0	0	0	0	1	0	0	0	0
45回目	p_{10}	0	1	1	1	1	1	1	1	0	0	0
46回目	p_9	1	1	1	1	1	1	1	0	0	0	0

図 7　タップルの更新パターン

試行	ダミー数
trial 1	7
trial 2	7
trial 3	43
trial 4	8
trial 5	4
trial 6	17
trial 7	2
trial 8	0
trial 9	0
trial 10	30
sum	118

(a)ダミー数合計

	下限値	上限値	タップル数	ダミー数	ダミー数を除いたタップル数
p[1]	0	8648	8	0	8
p[2]	8648	10304	8	0	8
p[3]	10304	13693	8	0	8
p[4]	13693	17010	7	0	7
p[5]	17010	20750	7	0	7
p[6]	20750	34779	7	0	7
p[7]	34779	27481	7	0	7
p[8]	27481	30948	7	3	4
p[9]	30948	33360	7	1	6
p[10]	33360	36300	7	1	6
p[11]	36300	37552	7	3	4
p[12]	37552	38763	7	3	4
p[13]	38763	42640	7	1	6
p[14]	42640	43184	7	5	2
p[15]	43184	46012	7	3	4
p[16]	46012	46579	7	5	2
p[17]	46579	48591	7	1	6
p[18]	48591	49556	7	6	1
p[19]	49556	49627	7	6	1
p[20]	49627	50000	7	5	2
				43	

(b) trial 3における各バケットのダミー数分布

図 8　ダミーデータ

4.3 解決方法

前節に述べた各問題点に対する解決方法の方針を以下に示す．

バケット調整にかかる時間の短縮　各バケット内のヒストグラムを索引情報に格納し，それを用いてバケット調整を行うことによりサーバからのデータ取得および復号化にかかる処理を減らす．

隣接関係が推測されないバケット更新　バケット更新時に，更新しないバケットのうち該当タップル数が最大でないバケットにダミータップルを挿入する．

ダミータップル数の削減　前節に述べたようにダミータップルが増えてしまうのはタップルを挿入するたびに差分的にバケットの調整を行っていることが一つの原因として考えられる．削減法としては，バケット数や該当タップル数の差 MaxDiff の動的な調整，また，定期的にバケット分割の組み直しなどを行うことによってダミータップル数を削減する方法が考えられる．

これらの解決法のうちヒストグラムを導入した改良法について次節に説明しその検証を行う．(他の解決法については今後の課題とする．)

4.3.1 ヒストグラムを導入した索引構成法

実験結果より，実行時間にはサーバから取得するデータ量が非常に影響することが分かった．よって，サーバから取得するデータの量を最小限に抑えるため，3 章で提案した索引構成法にさらにヒストグラムを導入する．

[ヒストグラムの生成]

図 9 に示すように，それぞれのバケット内を等間隔の bin に分割する．bin の幅はあらかじめ決めておくこととする．それぞれの bin に該当するタップル数を文字列にし，属性”histogram”を索引情報に付加し文字列を格納する．図 9 の例では，属性”histogram”の値は”0:3,1:7,3:2,4:7,5:5” というように左から順にヒストグラム番号を付け，ヒストグラム番号と該当タップル数のペアを並べる．

[ヒストグラムを用いたバケット調整]

正の方向に切り崩しを行う場合，切り崩しを行うバケット p_i 内で 2 番目に大きい（右から 2 番目の）bin の最大値をバケット p_i と p_{i+1} の新しい境界とする．また負の方向に切り崩しを行う場合，切り崩しを行うバケット p_i 内で最小の（1 番左の）

bin の最大値をバケット p_i と p_{i-1} の新しい境界とする．図 10 は負の方向への切り崩しの例を示したものである．

このように，ヒストグラムを利用してバケット調整を行うことにより，新しい境界線を決定するためにサーバからデータを取得する必要がなくなる．ヒストグラムの情報は索引情報に格納されているので，索引情報を復号化し見るだけでバケット調整ができる．また 1 個のデータ挿入の際に行われるバケット調整は 1 回とは限らず，複数回ある場合がある．このとき 1 回の調整の度にサーバ上にある暗号化テーブルを更新するのでは無駄な時間がかかってしまう．調整の度に，ヒストグラム情報を含む索引情報だけを更新し，バケットの範囲変更の情報は蓄積していく．そして全ての調整が終了してから暗号化テーブルを更新する．そうすることで，更新タップル数も最小限に抑えることが出来る．

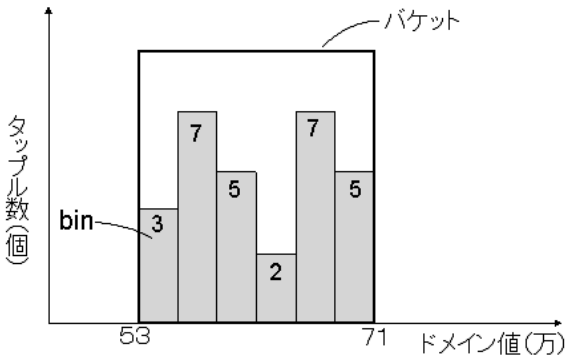


図 9　ヒストグラム生成

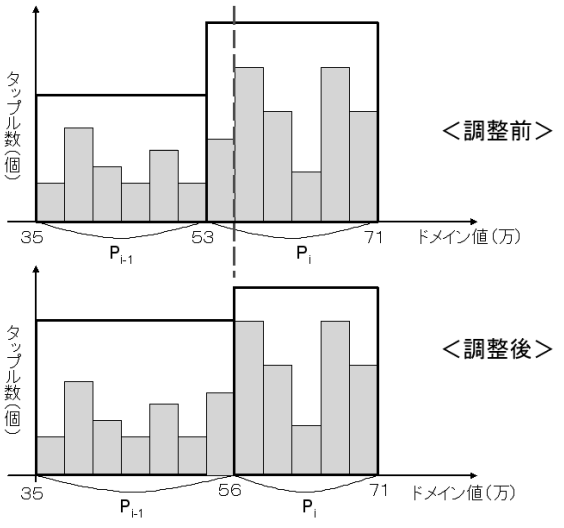


図 10　ヒストグラムを用いたバケット調整

4.3.2　改良法の検証

ヒストグラムを導入した索引構成法において，ヒストグラム導入前の実験と同じ条件でデータ挿入を行い，その実行時間を計測した．ただし，バケット内の bin の幅は 20 とした．図 11 はヒストグラム導入後の実行時間 (a)，取得タップル数 (b)，更新タップル数 (c) である．

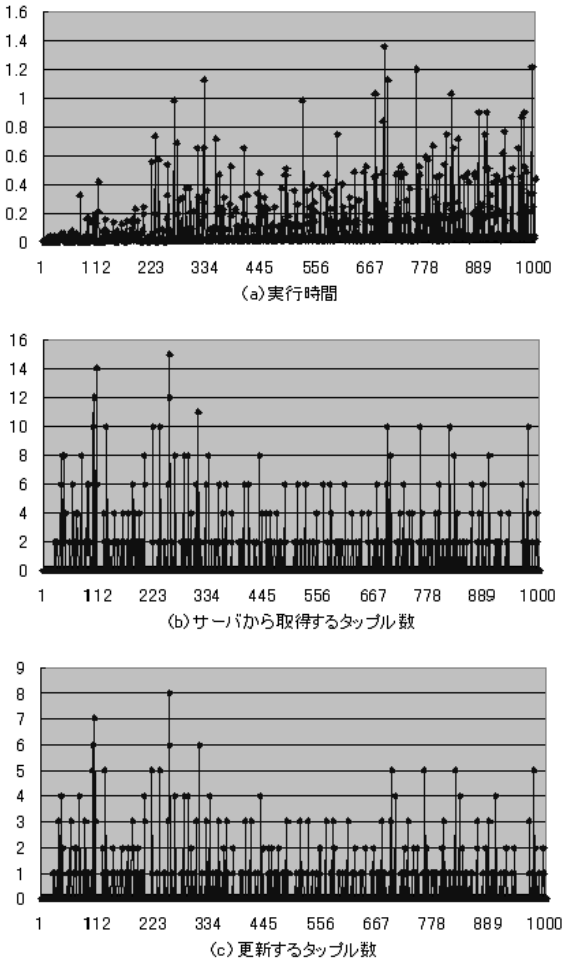


図 11　ヒストグラム導入後の実験結果 (a) 実行時間, (b) サーバから取得するタップル数, (c) 更新するタップル数

図 6 と比べてみると，ヒストグラム導入後はサーバからの取得タップル数と更新タップル数に大きな差がなくなっており，更新タップルに応じて必要な分だけのタップルがサーバから取得されるようになっていることが分かる．そのため，実行時間も大幅に短縮され，最大時間で比べると 3 分の 1 から 4 分の 1 程度にまで小さくなっている．このことから，ヒストグラムの導入は実行時間短縮に有用であると考えられる．

5. 関 連 研 究

データベースにおけるプライバシー保護やセキュリティの研究は，従来よりデータベースの主要な研究分野として数多く行われてきた [3]．近年ではプライバシー漏洩の問題に対する推論制御，統計データベースなどが注目されている．

また，一部の情報を隠蔽したり暗号化することで十分にプライバシーを保護したまま問合せやデータマイニングをするための研究も盛んに行われている [8]．Agrawal らの提案した Reconstruction-BasedApproach [1] は個人情報に意図的にランダムノイズを乗せてプライバシーを保護し，ベイズの定理を用いて，逐次的に確率密度関数を推定し，決定木によるマイニングを実現する．また，k-anomymity [5] はデータを開示する際，一部の情報を抽象化したり隠蔽することで同じ情報を持つ人数を一定数以上に制御する手法である．

暗号化データベースも個人情報をサーバ管理者に対して保護しながら問合せを行う手法であり，暗号化したタップルに付随する索引を用いて 2 段階の検索を行い，問合せを実現している．本手法は文献 [4] によって提案され，その後集約演算に対する処理やアクセス制御への適用が研究されている．安全な索引構成法に関しては文献 [2] にて Equi-depth による分割法を利用し，処理効率と安全性を考慮した最適なバケット数を求める提案がされている．しかしながらこれらの研究ではデータはあらかじめすべて用意されているという前提であり，更新が起こることを想定していない．複数の利用者で暗号化データベースを共有する場合には更新が頻繁に行うこともあると考えられるため，我々の手法も必要となると考えられる．また適切なバケット数の導出法に関しても文献 [2] を参考に更新に対応できるよう改良する必要があると考えられる．

6. まとめと今後の課題

本稿では新しい索引構成法の 1 つとして，MaxDiff での領域分割法を提案した．本提案手法はバケットの個数を固定し，バケット調整により高さを均等化する方法をとっている．また，ヒストグラムを導入し 1 回のバケット調整の度にサーバ上の暗号化テーブルをスキャンする必要をなくし，計算量を小さくする方法を併せて提案し，その有用性を実験により確認した．

本稿で行った試みは安全な索引を構成するための初期的な試みであり，今後多くの課題が残っている．例えば，本提案手法は，年齢や月給等の特定の範囲のある数値データには適用できるが、住所、氏名のようなデータには効率よく適用できないため，範囲を持たないデータに対する対処法を考えなければならない．また，削除や更新に伴うバケット調整，適切なバケット数やバケット内タップル数の検証やそれらを動的に変更するアルゴリズムの検討も必要である．また，今回は 1 属性のみを扱ったがテーブル内にある複数の属性に対する方法も考えなければならない．また，今回の索引構成法は部分一致による検索に対処することができないことから，それらに対処する方法も考えていく必要がある．また，今回は提案手法の懸念事項の解決法としてヒストグラムを用いた更新時間短縮の検証のみを行ったが，ほかの解決法も試みる必要がある．また，本提案手法ではバケット調整による索引値の変化をデータ挿入ごとに観測した場合，バケットの隣接関係がわかってしまう．例えば，給与の索引値が 12234 であったタップルがテーブルの更新後に 1973 という索引値に変わった場合，バケット番号が 12234 のバケットと 1973 のバケットは隣接関係にあることが分かってしまう．これを解決するためには，

このように，どの更新がバケット調整により生じた真の更新かを分からなくすることで隣接関係が推測されないようにできると考えられるので，検討していきたい．

今後はこれらのシステムを実装，改良し，安全性やコストの面から比較・検証していきたい．

文　　献

[1] R. Agrawal and R. Srikant:. “Privacy-Preserving Data Mining,” *Proceedings of ACM SIGMOD Conference on Management of Data*, pp.439-450, 2000

[2] Hore B., Mehrotra S. and Tsudik G.: “A Privacy-Preserving Index for Range Query,” *Proceedings of the 30th VLDB Conference*, pp.720-730, 2004.

[3] Maria Grazia Fugini, Silvana Castano(Editor), Giancarlo Martella (Contributor): “Database Security”, *ACM Press Books/Addison-Wesley Publishing*, p.456, 1995.

[4] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra.: “Executing SQL over Encrypted Data in the Database-Service-Provider Model,” In *Proceeding of the 2002 ACM SIGMOD International Conference on Management of Data*, pp. 216–227, June 2002.

[5] Sweeney L.: “K-anonymity: A Model for Protecting Privacy,” *International Journal on Uncertainly, Fuzziness and Knowledge-based System*, 10(5), pp.557-570, 2002.

[6] G. Piatatetsky-Shapiro and C. Connell.: “Accurate estimation of the number of tuples satisfying a condition,” In *Proceeding of ACM SIGMOD*, 1984.

[7] V. Poosala, P. J. Haas, Y. E. Ioannidis and E. J. Shekita.: “Improved histograms for selectivity estimation of range predicates,” *Proceedings of ACM SIGMOD*, pp.294-305, 1996.

[8] Verykios V. S., Bertino E. and Fovino N.: “State-of-the-art in Pricacy Preserving Data Mining,” *SIGMOD Record*, 33(1), 2004.

[9] M. Winslett and J. D. Ullman.: “Jeffrey D. Ullman speaks out on the future of higher education, startups, database theory, and more,” *SIGMOD Record*, 30(3), 2001.