データベース・システム 74－1
（１９８９．１１．１７）

更新処理を考慮したリンク構造による物理スキーマ設計

*木實新一　　**古川哲也　　*上林彌彦

九州大学工学部*　九州大学大型計算機センター**

リンク構造を用いたデータベースの物理スキーマを，更新処理を考慮して設計する方法について述べる．リンク構造の設計では，構造に冗長性を付加することで検索処理を効率よく行うことが可能な構造を得ることができる．しかし，一般に，冗長度の高いリンク構造は更新処理オーバヘッドが大きくなる．本稿では，冗長な属性あるいは冗長なレコード型を更に付加して，検索処理の効率を落とすことなくこのオーバヘッドを抑える方法を提案する．この方法を用いれば，削除の最悪処理時間は，nをデータベースのサイズとしてO(n²)からO(n)に効率化可能である．

A LINK-STRUCUTURED PHYSICAL DATABASE DESIGN ALLOWING EFFICIENT UPDATES

*Shinichi Konomi　　**Tetsuya Furukawa　　*Yahiko Kambayashi

Faculty of Engineering, Kyushu University*　Computer Center, Kyushu University**

6-10-1 Hakozaki, Higashi, Fukuoka 812, Japan

In link-structured physical databases, the cost of data retrieval is often minimized through the introduction of redundancies in the link structure. Usually, however, the greater the redundancy, the greater the cost of maintaining it after an update. In this paper, we show how the addition of redundant attributes or redundant record types may be used to lower the overhead in processing updates, while still allowing for efficient data retrieval. In this manner, the usual $O(n^2)$ worst-case time cost for the deletion process may be reduced to $O(n)$ where n is the size of the database.

1．まえがき

リンク構造は，データ間の対応を構造にそのまま反映している．リンク構造においてリンクに沿ったデータ間の対応を求める際は，構造から直接対応が求まり，一般に処理効率がよい．ネットワークデータベースやオブジェクト指向データベースは，リンク構造を用いているため，関係データベースに比べて，一般にデータ間の対応を求める検索処理を効率よく行うことが可能である．また，関係データベースでは，結合処理効率化のため索引構造[1][2]を用いるが，これもリンク構造の一種であると考えられる．このように，データベースの処理効率化の点から，リンク構造の導入は本質的であると考えられる．効率よいデータベースの実現は，データモデルに関わらず物理スキーマをリンク構造にすることで可能となる．

リンク構造によるデータベースを設計する際に考慮すべきことは，従属性制約や検索，更新等の処理効率である．従属性制約は，これを設計時に考慮することで，データの冗長性を削減でき，更新時の制約保持が容易に可能となる．既存のデータベース設計法には，関数従属性[3]や多値従属性[4]などの従属性制約を考慮した方法，実体関連図等を用いた直感的な方法 [5][6][7]，検索処理を考慮した方法[8][9]などがある．[8][9]では，検索結果のデータ対応を冗長な構造に保持するスキーマを作る方法を用いている．一般に，検索処理を考慮したスキーマを設計する場合，リンク構造の冗長性が高くなる．更新処理効率は冗長性と深く関わっており，冗長性が高いほど更新処理コストは高くなる．

検索処理効率のよい冗長なリンク構造を設計した場合，冗長性により更新処理コストが高くなる．このため，検索と更新の処理効率のトレードオフを解決する必要が生じる．よって，検索効率化は，特に頻繁であるような検索処理のみを対象とし，不必要に冗長性を高くしないようにする必要がある．それのみでなく，更にリンク構造を冗長にすることで，この様な冗長性に起因する更新処理オーバヘッドを検索処理の効率を悪化させることなく抑えることもできる．図１はこれを概念的に示している．スキーマS_0を検索処理の効率がよくなるように変換するとS_1のスキーマになる．S_1に更に冗長性を付加するとS_2を設計することができる．

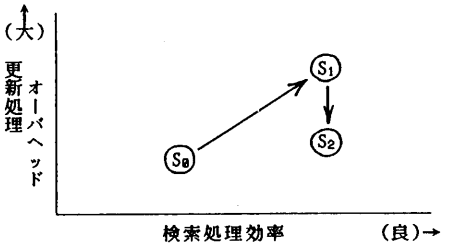


図１　検索処理効率と更新処理オーバヘッド

データの一貫性を保つために，冗長な部分に対応する部分のデータ更新を反映させる際には，冗長な部分を書き替える処理以外に，あるデータが冗長な部分のどのデータに対応しているか調べるための検索処理が必要となる．また，複数のデータが冗長な部分の１つのデータに対応している場合は，データの更新を冗長な部分に反映させないことがあり，常に冗長な部分を更新するとは限らないため，冗長な部分に更新を反映させるかどうか決定するための別な検索処理も必要となって来る．本稿では，リンク構造のキーを含むレコード型を付加してスキーマがキー包含条件を満たすようにするか，いくつのデータが冗長な部分の１つのデータに対応しているかを数え上げて冗長な部分に保持しておくか，いずれかの方法により更新時の処理として必要な検索が効率よく行えるスキーマを設計する方法を提案し，更新処理効率を評価している．更新処理効率は，これらの方法を用いることにより，nをデータベースのサイズとして，削除の際の最悪で$O(n^2)$の処理時間が$O(n)$に効率化可能である．

2．基本的事項

データベースにおけるリンク構造として代表的なものにネットワーク構造がある．ネットワーク構造（ネットワークスキーマ）は，バックマン線図と呼ばれる有向グラフB(V,E)で表される．Vはレコード型に対応する節点の集合，Eは親子集合型に対応する枝の集合である．図２はバックマン線図とその実現値の例である．

ここで，R(X)は属性集合Xからなるレコード型Rである．また，下線の属性集合はレコード型のキーであり，その値を定めれば各レコード型でレコードが一意に定

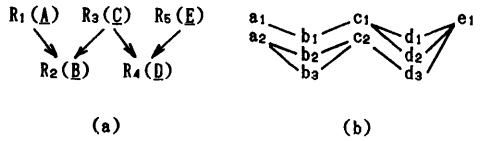


図２　ネットワークスキーマとその実現値

まる．値を定めればＢの全てのレコード型でレコードが一意に定まるような属性集合をＢのキーと呼びK(Ｂ)で表す．一般にK(Ｂ)はＢ中の子レコード型を持たないレコード型のキー集合の部分集合であり，図１(a)のキーはBDである．

ネットワーク構造に冗長なレコード型や親子集合型を加えたり，レコード型に冗長な属性を加えたりすることで，検索処理効率の良いネットワーク構造が得られる[8][9]（図３）．

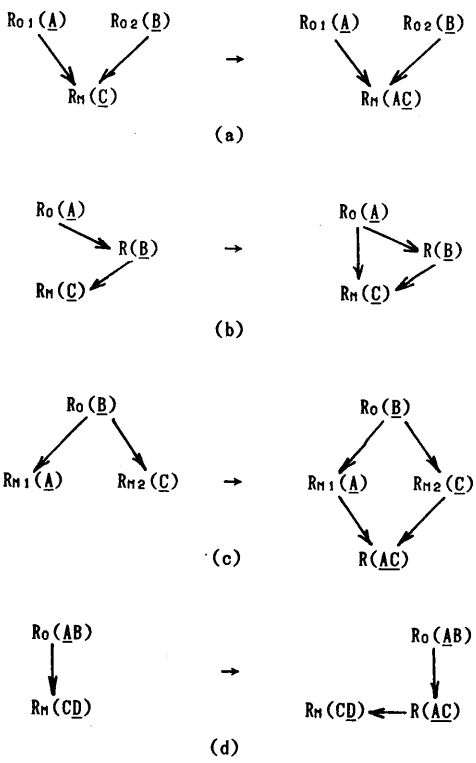


図３　スキーマ変換の基本操作

(a) RO1の属性AをRMに付加する．ABの対応はRO2,RMで求めることができる．

(b) ROとRM間にRを介するレコードの対応を表す親子集合型を付加する．ACの対応はRを検索しなくても求められ，検索するレコード型が削減される．

(c) RM1,RO,RM2の経路でのACの対応を表すレコード型R(AC)を付加する．ACの対応はRで直接得られる．

(d) ROとRMとの間にレコード型R(AC)を加える．ABCの値はR,ROで求めることができ，この部分のキーはACとなるため解が重複しない．

効率よい処理が必要な検索についてのみネットワーク構造に冗長性を付加し，効率化を行う．

[例１]　図４は，属性Aと属性Eの値の対応が効率よく求められるように図２を変換したものである．冗長な構造として，図２の構造が表す属性Aと属性Eの値の対応をレコードとして保持するレコード型R6(AE)と，それに付随する親子集合型を加えることで，検索処理を効率化している．図４では，レコード型R1からR6までを見なくてもR6のみで属性AとEの対応を求めることができる．

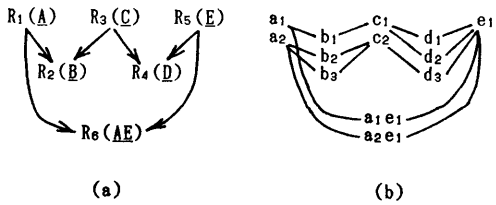


図４　検索処理を考慮したスキーマとその実現値

３．更新処理の複雑さのクラス

ネットワーク構造での更新は，次のように分類することができる．

(1) レコードの連結性を変えない更新．
　属性値の変更など．
(2) レコードの連結性を変える更新．
　(2-1) レコードの挿入、リンクの付加．
　(2-2) レコードの削除、リンクの削除．
　(2-3) リンクの変更（つけかえ）．（リンク付加と削除を組み合わせたものとして考えることができる．）

ネットワーク構造Bに冗長性がなければ，データの一貫性を保つための処理の手間が省け，一般に上記の各更新処理は容易となる．しかし，BにBから得られるデータを表す冗長な構造B'が付加された場合，更新処理は複雑になる．これは，データの一貫性保持のため，BとB'で対応するデータ同志に矛盾が生じないようにB中の更新をB'中に反映させねばならないためである．B中の更新をB'中に反映させるためには次のような処理が必要である．

(1) 被更新データ候補検索：B中の更新するデータがB'中のどのデータと対応しているかを調べるための検索処理
(2) 更新決定検索：(1)で求められたB'中のデータにB中の更新を反映させるか決定するための検索処理．
(3) 実更新：(2)での結果に応じて，(1)で求められたB'中のデータを実際に書き替える．

図5に冗長性のあるスキーマで更新時に必要な処理を示す．

- 更新処理
  - Bでの更新処理
  - B'への更新反映処理
    - 被更新データ候補検索
    - 更新決定検索
    - 実更新

図5　冗長性のあるスキーマにおける更新時の処理

なお，以下ではBに閉路が存在しない場合のみを考察の対象としている．

[例2] 図4での更新処理を考える．R_6及びそれに付随する親子集合型をB'，その他のレコード型及び親子集合型をBとする．Bでレコードb_2を削除する場合を考える．属性AとEに関して，b_2につながるものを検索すると，a_2とe_1の対応が求められる（被更新データ候補検索）．しかし，他の経路でa_2とe_1がつながっているかも知れないため，a_2からe_1あるいはe_1からa_2まで再び検索を行う必要がある（更新決定検索）．この場合，b_2を介するリンクの経路以外にb_3を介するリンクの経路が存在するためB'中のレコードa_2e_1は削除されない．□

一般に，更新反映の処理では，実更新の前に被更新データ候補検索及び更新決定検索を行う必要があるが，次のようなレコードの更新においては被更新データ候補検索が不必要となり更新処理コストが低くなる．

[定義1] 更新するレコード型がその更新を反映することが必要なレコード型の祖先である場合の更新反映を遺伝的更新反映と呼ぶ．□

[補題1] 遺伝的更新反映では更新決定検索が不要である．□

(証明) 遺伝的更新反映では，レコード型の親子関係から，更新が行われるレコード型のレコードとその更新が反映されるレコード型のレコードとがリンクで1対多に対応づけられていることが分かる．従って，更新が反映されるレコード型の1つのレコードは，更新が行われるレコード型の1つのレコードとしか対応づけられていない．更新の反映は対応するレコードの更新に応じて行うが，対応するものが1つしか無ければ更新は必ず反映されることが保証されており，更新反映を行うかどうかを検索によって決定する必要が無い．(証明終り)

[例3] 図4では，B中のレコード型$R_1,\cdots,R_5$のレコードの更新をB'中のレコード型R_6に反映させる必要がある．このうちレコード型R_1,R_5はR_6の親レコード型であり，更新反映は遺伝的更新反映である．B中でレコードR_1のa_1を削除する場合を考える．B'中のレコード型R_6でこのとき削除すべきレコードa_1e_1は，B中で削除するレコードa_1から直接リンクをたどることで求められる（被更新データ候補検索）．求められたレコードa_1e_1を削除する（実更新）．□

遺伝的更新反映では，属性値の変更や削除において，B中の更新するレコードにつながるB'中のレコードのデータをそのまま書き替えればよい．挿入では，Bでの新たなレコードのつながりは，必ずB'中の新たなデータとなるため，検索をしてそれを確かめる必要がない．

一般のスキーマでは，B中のどのレコード型で更新

が行われるかによって遺伝的更新反映であるかどうかが決ってくる．しかし，スキーマが次の条件を満たしていれば，属性値の変更については常に更新反映が遺伝的更新反映となる．

［定義２］ 更新属性連結条件：B中のレコード型R_i（$1 \leqq i \leqq n$）とB'中のレコード型（$1 \leqq j \leqq m$）に共通属性があるとき，R_iとR_jを結ぶ親子集合型が存在する．

［補題２］ 更新属性連結条件が満たされるスキーマでの属性値の変更では常に更新反映は遺伝的更新反映である．□

（証明） 属性値の変更では，B中のレコード型とB'中のレコード型とで共通属性をもつもの同志のみについて更新を反映し，データの一貫性を保たねばならない．更新属性連結条件が満たされるスキーマでは，このような共通属性をもつB中とB'中のレコード型同志が親子集合型で直接結ばれており，これらのレコード型間では親レコード型から子レコード型へと更新を反映できる．よって，属性値の変更では，更新属性連結条件が満たされるスキーマでは更新反映が常に遺伝的更新反映となる．（証明終り）

[8][9]で提案されている設計法を用いると，更新属性連結条件が満たされるスキーマが一般に設計される．以下では更新属性連結条件を満たすスキーマであることを仮定して議論を行う．

挿入や削除では，B中でB'中の属性集合を共通属性をもたないレコード型の更新もB'中に反映されることがあるため，更新属性連結条件が満たされても，Bで更新するレコード型によってはB'への更新反映が遺伝的更新反映とならないことがある．しかし，挿入，削除についてもスキーマが次の条件を満たしていれば，B中のどのレコード型で更新が行われてもB'への更新反映は常に遺伝的更新反映となり，更新決定検索が不必要となり効率よい更新処理が可能となる．

［定義３］ キー包含条件：冗長な構造B'には，属性集合がBの対応する部分の超キー（キーの超集合）となるレコード型が存在する．

［補題３］ キー包含条件を満たすスキーマで，B中の各レコード型の更新は，B'中のキーを包含するレコード型へ遺伝的に更新反映される．□

（証明） キー包含条件を満たすスキーマでは，B'のキーを包含するレコード型の属性からB中の各レコード型の属性へ関数従属性がある．このため，B中のどのレコード型もB'のキーを包含するレコード型の祖先となり，BからB'に遺伝的に更新反映することが可能である．（証明終り）

［定義４］ 更新が行われるレコード型がその更新を反映することが必要なレコード型の親または子レコード型である場合の更新反映を直接的更新反映と呼ぶ．□

［補題４］ 直接的更新反映では削除での更新決定検索が不要である．

（証明） 削除が行われるレコード型がその削除を反映することが必要なレコード型の親レコード型であれば，遺伝的更新反映であり，補題１より更新決定検索は不必要である．逆に，子レコード型である場合，削除が行われるレコードにつながる更新反映に必要なレコード型のレコードをまず被更新データ候補として検索する．求められたレコードは，そのレコードと更新が行われるレコード型の他のレコードがリンクでつながっていない場合削除し，その他の場合削除しない．この場合のリンクの有無は，被更新データ候補検索で求められたレコードのポインタから判断できるため，検索せずに更新決定を行える．よって，直接的更新反映では削除での更新決定検索が不必要である．（証明終り）

［定理１］ キー包含条件を満たすスキーマにおける更新処理では，更新決定検索が不要である．□

（証明） 補題３より，B'中のキーを包含するレコード型への更新反映は遺伝的更新反映である．よって，補題１より更新決定検索が必要ない．B'のその他のレコード型のレコードは，B'中のキーを包含するレコード型のレコードと１対多に対応しており，両者のレコード型間に親子集合型を付加するものとする．B'中のキーを包含するレコード型以外では，B'中の

キーを包含するレコード型の更新を反映させることで，直接的更新反映が行える．よって，補題４より，削除での更新決定検索は不用である．挿入については，B'中のキーを包含するレコード型の更新を反映させず，B中のB'中にリンクでつながるレコードのポインタの有無で更新決定を行える．B中のB'中にリンクでつながるレコードは被更新データ候補検索でアクセスされるので，このとき同時に更新決定を行えるため更新決定検索は不要である．属性値の変更では，更新属性連結条件が満たされるように適当な親子集合型を付加していれば，補題２より常に遺伝的更新反映となる．よって，補題１により更新決定検索が不要となる．以上よりキー包含条件を満たすスキーマでは，更新決定検索をが不要となる．(証明終り)

[例４] 図４はキー包含条件を満たさないスキーマの例であり，図２のスキーマ（キーはBD）に，冗長なレコード型R_7(BD)を加えた場合（図５）は，キー包含条件を満たす．B'中のキーを包含するレコード型R_7(BD)への更新反映は遺伝的更新反映となり，レコード型R_1～R_5のレコードの更新は，つながるR_7のレコードを求め，そのまま更新反映させればよい．

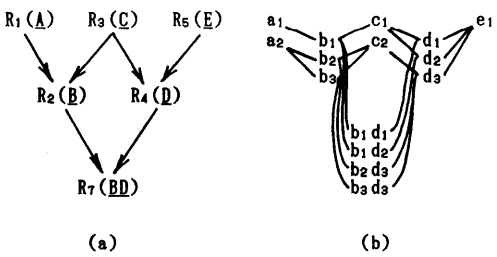


図５　キー包含条件を満たすスキーマ

４．更新処理を考慮したスキーマの設計

キー包含条件を満たさないスキーマにBのキーを包含するレコード型を付加して，条件を満たすスキーマを設計する方法を示す．

[手続き１]
(1) 従属性制約等により設計した冗長性の少ないスキーマをBとする．
(2) Bに冗長な構造B'を付加して，特定の必要な検索処理を効率化する．
(3) B'にBのキーを包含するレコード型Rを付加し，B'中のR以外のレコード型およびB中の子レコード型をもたないレコード型をRの親レコード型とする．

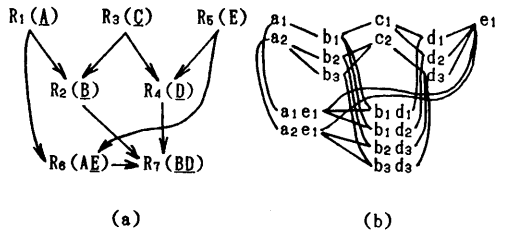
[定理２] 手続き１により設計されるスキーマにおける更新処理では更新決定検索が不要である．□

(証明) 設計結果のスキーマではB'中にBのキーを包含するレコード型R が含まれるので，キー包含条件を満たす．よって，定理１より更新決定検索が不要である．B中の子レコード型をもたないレコード型をRの親レコード型とすることでB中のレコード型はすべてRの祖先になり，B'中のR以外のレコード型はすべてR の親レコード型としているのでこの場合更新決定検索は不要である．(証明終り)

[定理３] 手続き１のステップ３は手続き１のステップ２で効率化した検索処理の効率（アクセスを要するレコード数）を悪化させない．□

(証明) 手続き１において，ステップ３ではステップ２での構造をそのまま残して更に冗長な構造を付加している．効率化した検索の処理はステップ３で付加した部分を除いた部分で処理が可能であり，ステップ３により効率が悪化することはない．(証明終り)

[例５] 図４(a)のスキーマは，属性A,Eの値の対応を求める検索の効率はよいが，キー包含条件を満たさない．これにR_7を付加した図６(a)のスキーマは，キー包含条件を満たし，R_7への更新反映は遺伝的更新反映である．削除の際のR_6の更新はR_7の更新を反映させることで局所的に行うことができる．即ち，R_7のレコードをまず削除し，そのレコードにつながるR_6のレコードがR_7の他のレコードとつながっていない場合のみR_6のレコードを削除する．挿入では，R_7のレコードはBで検索して求まったデータを即挿入し，R_6のレコードはBで検索して，R_1とR_5のレコードのR_6へのポインタが無ければ挿入する．このように，B'への更新反映は更新決定検索が不要であり，容易である．属性A，Eの対応を求める検索はR_6で図４と同様に行える．□



(a) (b)

図6　更新処理と検索処理を考慮したスキーマ

5．数え上げ法を用いた更新処理の効率化

キー包含条件を満たすスキーマの削除において，B'中のキーを包含しないレコード型のレコードはリンクでつながるキーを包含するレコード型のレコードの数により実際に更新するかどうかを決定することができた．この，キーを包含するレコード型のレコードの数を数値で表し，B'中のレコード型の新たな属性の値として数え上げても効率よい更新処理が可能である．このうような数え上げ法を用いた更新処理の効率化が可能なスキーマを設計する方法を示す．

[手続き2]

(1) 従属性制約等により設計した冗長性の少ないスキーマをBとする．

(2) Bに冗長な構造B'を付加して，特定の必要な検索処理を効率化する．

(3) B'の各レコード型に数え上げのための属性Nを付加する．N の値は，そのレコードが対応するB中のキー値数を示す．

[定理4] 手続き2により設計されるスキーマにおける更新処理では，更新決定検索が不要である．□

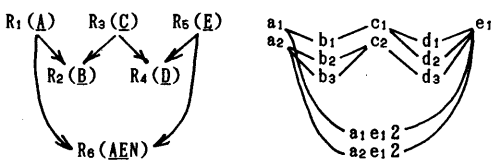
(証明) 属性値の変更では，更新属性連結条件が満たされるように適当な親子集合型を付加していれば，補題2より常に遺伝的更新反映となる．よって，補題1により更新決定検索が不要となる．挿入については，B中のB'中にリンクでつながるレコードのポインタの有無で更新決定を行える．B中のB'中にリンクでつながるレコードは被更新データ候補検索でアクセスされるので，このとき同時に更新決定を行えるため更新決定検索は不要である．削除については，N の値により更新決定を行える．ので更新決定検索は不要である．(証明終り)

手続き2により設計されるスキーマでは，更新決定検索は不要であるが，Nの値の維持がやや複雑となる．Nの値はB'中のレコードがB中の何種類のリンクの経路とつながっているか調べれば求めることができる．挿入や削除では毎回新しくできたり失われたりするB中のリンクの経路が何集類あるか調べてN の値を書き替える必要がある．挿入や削除ではこのため，毎回B全体を被更新データ候補検索で調べる必要がある．

[定理5] 手続き2のステップ3は，手続き2のステップ2で効率化した検索処理の効率（アクセスを要するレコード数）を悪化させない．□

(証明) 手続き2において，ステップ3ではステップ2での構造をそのまま残して更に冗長な属性を付加している．ステップ3ではB'中のレコードのサイズが冗長属性の分やや大きくなるだけで，ネットワーク構造自体には変化が無い．このため，ステップ3の適用後に，ステップ2で効率化した検索処理でアクセスするレコード数に変化はない．(証明終り)

[例6] 図6のキー包含条件を満たすネットワーク構造に対応する数え上げ法の実現を図7に示す．ここで，属性Nの値は数え上げ値である．ここで例えばa_1e_1は$a_1b_1c_1d_1e_1$と$a_1b_1c_1d_2e_1$を通る2つのリンクの経路とつながっているからNの値は2となる．



(a) (b)

図7　数え上げ法の実現

このスキーマでの更新処理では，$R_1,\cdots,R_5$の挿入に対し，挿入したレコードにつながるリンクの経路を数

え上げ，すでにデータが存在すれば数え上げ値を増加させ，新たなデータの場合はそのデータを数え上げ値とともに挿入する．削除に対しては，削除するレコードにつながるリンクの経路を数え上げて，数え上げ値をその分減少させ，値が０となれば実際にデータも削除する．属性値の変更は，遺伝的更新反映の条件を満たすため，リンクでつながるレコードの対応する値を書き替えるだけですむ．

６．更新処理コストの比較

Ｂを冗長性のないスキーマ，Ｂ'をＢの更新を反映する冗長なスキーマであるとする．Ｂ中のレコード型 $R_i \in V$ $(1 \leq i \leq n)$ の各レコードの更新処理をすべて１度ずつ行ったとき合計でアクセスされるレコード数の最悪の値をR_iのレコード数で平均した値（レコード数）で更新処理コストを，レコードの挿入，削除，属性値の変更に分けて評価した（それぞれ Ins,Del,Modとする）．ここで，最悪はスキーマの実現値についての最悪であり，平均はR_iで更新するレコードについての平均である．R_iのレコードを更新する際は，まずR_iのレコードにアクセスするが，このレコード数は評価に加えていない．評価の対象としているのは，R_iのレコードの更新を冗長な部分Ｂ' に反映させるための検索（被更新データ候補検索及び更新決定検索）でアクセスされるレコードの数及び実更新でアクセスされるレコードの数である．また，R_iの各レコードを１度ずつ更新する際は，あるレコードの更新を考えた後，そのレコードが更新される前の状態に戻して，次のレコードの更新を考えるものとする．

[定義５] $|R_i|$をレコード型R_iのレコードの数，$|B|$をＢ中のキーのデータ数，RN(B)，RN(B')をＢ，Ｂ'中のレコード型の数，$\|B'\|$をＢ'中の全レコード型のレコード数の総和とする．□

[定理６] キー包含条件を満たさないスキーマでは，

$$Ins=(|B|/|R_i|)*(RN(B)-1)+\|B'\|/|R_i|$$

$$Del=(|B|/|R_i|)*(|B|-1)*RN(B)+(|B|/|R_i|)*(RN(B)-1)+\|B'\|/|R_i|$$

$$Mod=\|B'\|/|R_i| \quad or \quad 0$$

□

（証明） スキーマがキー包含条件を満たさない場合，遺伝的更新反映とならない更新反映を行わねばならない．挿入については，Ｂ中のＢ' 中にリンクでつながるレコードのポインタの有無で更新決定を行える．Ｂ中のＢ' 中にリンクでつながるレコードは被更新データ候補検索でアクセスされるので，このとき同時に更新決定を行えるため更新決定検索は不要である．よって，被更新データ候補検索と実更新でアクセスされるレコード数を考慮すればよい．R_iの１つのレコードは平均Ｂの$|B|/|R_i|$個のキーに対応している．よって，被更新データ候補検索において１つのレコード型でアクセスされる最大のレコード数は平均 $|B|/|R_i|$個である．Ｂ中にはR_i以外にRN(Ｂ)-1個のレコード型が存在し，被更新データ候補検索ではそれら全てでレコードをアクセスするから，最大で平均$(|B|/|R_i|)*(RN(B)-1)$個のレコードにアクセスせねばならない．実更新については，平均$\|B'\|/|R_i|$個のレコードが挿入されるから，それらを加え合わせると

$$Ins=(|B|/|R_i|)*(RN(B)-1)+\|B'\|/|R_i|$$

削除については，被更新データ候補検索，更新決定検索，実更新が必要となる．被更新データ候補検索は挿入のときと同様であり，最大で平均$|B|*(RN(B)-1)$個のレコードにアクセスせねばならない．更新決定検索はR_iの１レコードの更新の被更新データ候補検索で得られる結果の最大で平均 $|B|/|R_i|$個データに対して，それぞれ，１つのレコード型あたり最大で$|B|-1$個のレコードを，Ｂ中の全レコード型（RN(Ｂ)個）について検索し，最大 $(|B|/|R_i|)*(|B|-1)*RN(B)$個のレコードにアクセスせねばならない．実更新では最大で平均$\|B'\|/|R_i|$個のデータを削除しなければならない．よって，

$$Del(B)=(|B|/|R_i|)*(|B|-1)*RN(B)+(|B|/|R_i|)*(RN(B)-1)+\|B'\|/|R_i|$$

属性値の変更については，更新属性連結条件を満たすように親子集合型を付加していれば，R_iのレコード型以外でアクセスするのは実更新する平均$\|B'\|/|R_i|$個のデータのみでよい．よって，

$$Mod=\|B'\|/|R_i|$$

ただし，R_iがB'中の属性と共通属性を持たなければ，更新反映は必要ないので，その場合，

$$Mod = 0$$

(証明終り)

[定理7] 手続き1により設計したキー包含条件を満たすスキーマでは，

$$Ins = (|B|/|R_i|)*(RN(B)-1)+\|B'\|/|R_i|$$
$$Del = (|B|/|R_i|)*(RN(B)-1)$$
$$\quad +(|B|/|R_i|)*RN(B')$$
$$Mod = \|B'\|/|R_i| \quad or \quad 0$$

□

(証明) 定理2より更新決定検索は不要であり，被更新データ候補検索と実更新のみ考えればよい．挿入及び属性値の変更については定理6と同様である．ただし，キーを包含するレコード型の新たな付加により$\|B'\|$が増加している．削除については，被更新データ候補検索で，定理6の場合と同様に最大で平均$(|B|/|R_i|)*(RN(B)-1)$個のレコードにアクセスする．削除の実更新については，R_iの1レコードが更新された時，まずキーを包含するレコード型で，平均$|B|/|R_i|$個のレコードが実更新される．それらのレコードそれぞれについて，キーを包含するレコード型と直接リンクでつながっている他の$RN(B')-1$個のB'中のレコード型で，リンクによりつながるレコードを求める．キーを包含するレコード型のレコードにつながる他のレコードは1つであるから，$|B|/|R_i|+(|B|/|R_i|)*(RN(B')-1)$個のレコードにアクセスする．よって，

$$Del(B) = (|B|/|R_i|)*(RN(B)-1)$$
$$\quad +(|B|/|R_i|)*RN(B')$$

(証明終り)

[定理8] 数え上げ法を用いて更新処理を行う場合，

$$Ins = (|B|/|R_i|)*(RN(B)-1)$$
$$\quad +(|B|/|R_i|)*RN(B')$$
$$Del = (|B|/|R_i|)*(RN(B)-1)$$
$$\quad +(|B|/|R_i|)*RN(B')$$
$$Mod = \|B'\|/|R_i| \quad or \quad 0$$

(証明) 定理4より，更新決定検索は不要であるから，被更新データ候補検索と実更新についてのみ考えればよい．属性値の変更については定理6と同様である．挿入，削除の被更新データ候補検索は定理6の場合と同様に$(|B|/|R_i|)*(RN(B)-1)$個のレコードにアクセスしなければならない．挿入，削除の実更新では，被更新データ候補検索で求められた最大でR_iの1レコードあたり平均$|B|/|R_i|$個のデータに対してB'中のレコード型の数え上げ値を書き替える必要があり，$(|B|/|R_i|)*RN(B')$個のデータに最大でアクセスしなければならない．よって，

$$Ins = (|B|/|R_i|)*(RN(B)-1)+(|B|/|R_i|)*RN(B')$$
$$Del = (|B|/|R_i|)*(RN(B)-1)+(|B|/|R_i|)*RN(B')$$

(証明終り)

定理6より，キー包含条件を満たさないスキーマでの更新処理の最悪時間コストは，nをデータベースのサイズ（ネットワーク構造のキーのデータ数$|B|$）として，$O(n^2)$である．ただし，処理時間はアクセスするレコード数に比例するものとする．また，nはレコード型の数より十分大きいものとして，レコード型の数$RN(B)$及び$RN(B')$を定数と見なし，$\|B'\|$については$\|B'\| \leq |B|*RN(B')$である．

これに対し，手続き1及び手続き2により設計したスキーマでは，定理7及び定理8により，更新の最悪時間コストは，$O(n)$である．手続き1により設計したスキーマと手続き2により設計したスキーマの違いは，手続き1で設計したスキーマでは遺伝的更新反映が常に可能であるのに対し手続き2により設計したスキーマでは常にそうとは限らないことである．遺伝的更新反映でない場合，被更新データ候補検索は，B中のB'中に含まれる属性を含むレコード型すべてを検索しなければならない．しかし，遺伝的更新反映であれば，B中の更新するレコード型のB'中の子孫へつながるB中のレコード型のみ検索すればよいため，手続き1により設計したスキーマでは，手続き2により設計したスキーマよりも被更新データ候補検索を局所的に行うことができる．このため，元の構造Bが複雑な場合，手続き1で設計されるスキーマの方が効率よい更新処理が可能となってくる．

［例７］　キー包含条件を満たす図６において，R_2での削除を考えると，Ｂ（R_1,…,R_5のレコード型及びR_6,R_7に付随しない親子集合型）中では検索が必要なく，リンクでつながるR_6とR_7のレコードに更新を反映させれば良い．数え上げ方を用いた図７で同様にR_2での削除を考えると，Ｂで全てのレコード型を検索する必要があり，求められた属性A,Eの値に基づいてＢ'中のデータにアクセスし，重複度の書き替えかレコードの削除を行う．キー包含を満たすネットワーク構造における更新処理の局所性のため，図６のスキーマの方が更新処理効率がよい．□

８．むすび

キー包含条件を満たすネットワーク構造及び数え上げ法を用いたネットワーク構造を設計し，冗長な構造への更新処理反映を効率よく行う方法を提案した．これらの方法を用いると，削除の際に特に複雑になる更新反映時の更新決定検索を行う必要がなくなり，更新処理の最悪時間コストは,nをデータベースのサイズとして，$O(n^2)$から$O(n)$に効率化可能である．また，この際，更新時の処理でない一般の検索処理の効率は悪くなることがない．

しかし，スキーマに冗長性がある限り，冗長な部分への更新反映オーバヘッドは皆無とならないため，本方法を用いてもなお検索処理と更新処理のトレードオフはなんらかの方法で解決する必要がある．また，キー包含条件を満たすネットワーク構造と数え上げ方を用いたネットワーク構造のうちどちらを用いるのがよいかを決定する方法についても，今後の考察が必要である．この際には，元のネットワーク構造の複雑さや，どの様な種類の更新処理が多く起こるか，更に，処理効率だけでなく記憶コストの点なども考慮しなければならない．シミュレーションによりネットワーク構造の設計結果を評価することも重要である．

参考文献

(1) Valduriez,P., "Join Indices", ACM Trans. on Database Syst., Vol.12, No.2, pp.218-246, June 1987.

(2) Bipin,C.D., "Performance of a Composite Attribute and Join Index", IEEE Trans. on Software Eng., Vol.15, No.2, pp.142-152, Feb. 1989.

(3) Kuck,S.M. and Sagiv,Y., "Designing Globally Consistent Network Schemas", Proc. ACM SIGMOD Int. Conf. on Management of Data, pp.185-195, May 1983.

(4) Lien,Y.E., "On the Equivalence of Database Models," J. ACM, Vol.29, No.2, pp.333-362, April 1982.

(5) Bouzeghoub,M., Gardarin,G. and Metais,E., "Database Design Tools: An Expert System Approach", Proc. Int. Conf. on Very Large Data Bases, Aug. 1985.

(6) Ferrara,F.M., "EASY ER: An Integrated System for the Design and Documentation of Data Base Applications", Proc. 4th Int. Conf. on Entity-Relationship Approach, Oct. 1985.

(7) Reiner,D., Brown,G. and Friedell,M., Lehman, J., McKee,R., Rheingans,P. and Rosenthal,A., "A Database Designer's Workbench", pp.347-390, 1987.

(8) 古川，上林，“質問処理効率化のためのネットワークデータベースのスキーマ変換”，電子情報通信学会論文誌，Vol.J71-D，No.10，pp.2111-2119，昭和63年10月．

(9) 古川，上林，木實，“ネットワークデータベース設計支援システムの開発”，情報処理学会研究報告，88-DBS-67-5，昭和63年9月．