32. 分散型データベース(JIPNET-DDB)における問合せ変換

(財)日本情報処理開発協会 滝沢 誠, 浜中栄治
Makoto Takizawa Eiji Hamanaka

1. 序

コンピュータネットワークを介して、地理的に分散しているデータベース(以降DBと略す)を結合し統合的に利用するシステムは、分散型データベース(DDB)と呼ばれるものである。DDBの開発には、大別して2つのアプローチがある。1つは、DDB全体に、ある仮想的なDBを設定し、この中身をある目的関数のもとで各サイトに最適配置する方法である。これは、分割型(top-down)DDBと呼ばれ、SDD-1[ROTHJ77], INGRES[STONM78]等がその例である。

もう1つのアプローチは、統合型(bottom-up)と呼ばれるものである。これは相異なる意味構造を持つ既存DBを統合し、DDB全体の統一視野をユーザに提供することを目的にしている。例として、POLYPHEME[ADIBM78]がある。

我々の目指すDDBも後者のタイプである。既に巨大なデータリソースが存在している事実を無視できないことと、更にリソースの最適配置機能を付加することによって容易に前者のアプローチへも適用できることの2点がその理由である。

本論文では、我々のDDBの全体アーキテクチャを述べるとともに、現在インプリメント中の問合せ変換について論じる。

2. 四層スキーマ構造(FSS)と統一アーキテクチャ

統合型DDB(以降単にDDBと呼ぶ)構築のために解決すべき問題点は、異種性問題、即ちDBMSのデータモデル、言語、機能、格納されたデータの意味構造、ホストコンピュータの相違によって生じる問題と、分散問題、即ちデータが地理的に分散していることによって生じる問題の2点である。これらを解決するための我々のアプローチを四層スキーマ構造(four-schema structure or FSS)[TAKIM78, 79a]と呼ぶ。

FSSでは、まず第1に各サイトのDBMSの異種性問題を解決する。これはE-Rモデル[CHENP76]によって、各サイトのスキーマの構文構造を共通記述することによって可能である。各サイトのスキーマを局所内部スキーマ(LIS)と呼び、共通記述されたものを局所概念スキーマ(LCS)と呼ぶ。LISからLCSへの変換過程を同種化と呼び、この過程で除去される異種性を、異種性情報(HI)[付記1]として各サイトが保持する。

次に、同種化された各サイトのLCSから、あるネットワーク共同体にとって意味のあるデータへ統合し、これを同じくE-Rモデルで記述する。これを全体概念スキーマ(GCS)と呼ぶ。GCSは、ユーザに対するDDB全体への統一的な視野であり、どこに何があるかという分散問題は現えなくなっている。LCSからGCSの生成過程を統合化と呼ぶ。統合化は、既存DBの意味構造から、新たな意味の生成過程であり、これは全体管理者(GA)により行なわれる。統合化で除去される情報(GCSとLCSとの対応情報)は分散情報(DI)と呼ぶ。

更に、共同体の各アプリケーションに必要なデータの意味を、これに適したデータモデルで記述したものを外部スキーマ(EXS)と呼ぶ。

第21回プログラミング
シンポジウム 1980. 1

この様にFSSは、4つのスキーマと、同種化及び統合化という2つの主要なスキーマ間のマッピングから成っている。後述する問合せ変換（QT）は、LCSからLISへの逆同種化であり、問合せ分割（QD）はGCSからLCSへの逆統合化である。FSSの全体アーキテクチャを、ANSI/SPARC[TSICD対]の記述法を用いて表わすと図2.2のようになる。

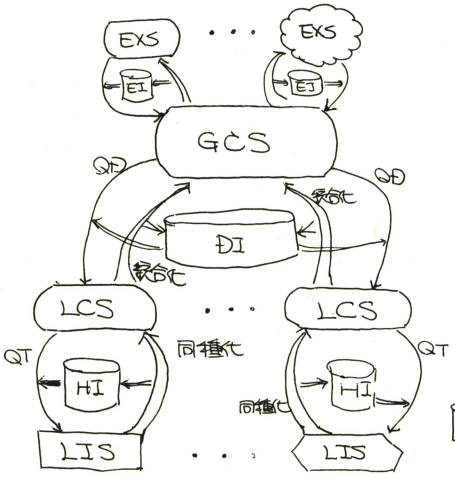

図2.1 FSS

NDD: network data directory
LA: local administrator
AA: application administrator

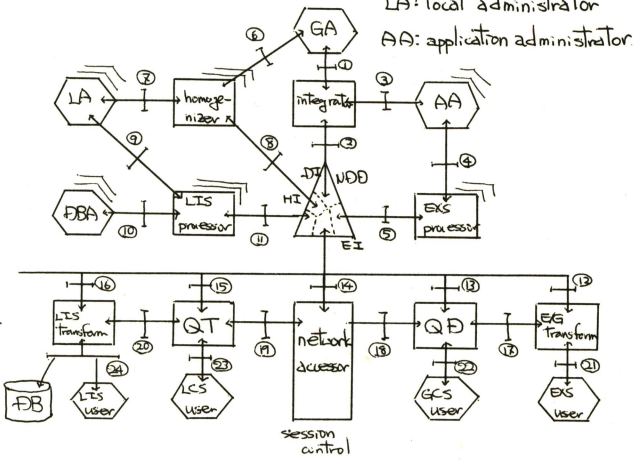

図2.2 統一アーキテクチャ

3. 同種化

同種化は、LISをE-Rモデルで共通記述することによって可能である。LCSレベルでは 共通言語としてQUEL[HELDG75]を用いる。同種化は表3.1を用いて、LIS要素を対応するE-Rモデル要素に1対1に置換することによって行なう。図3.2は、DBTG型のDBのスキーマ（図3.1）を同種化することによって得られたLCSを示している。ESRは、事象集合を表わすリレーションであり、RSRは関係性集合を表わすリレーションである。属性についた下線はこれがキーであることを、リレーションの下線はこれがスキームであることを各々示している。

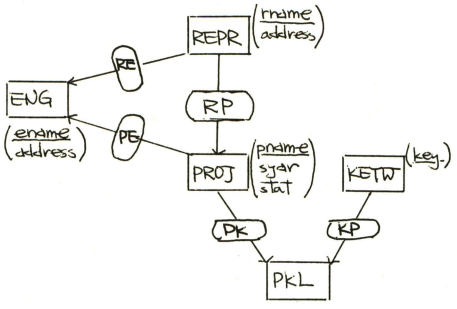

図3.1 DBTGタイプのLIS

ESR REPR (rname, address)
ENG (ename, address)
PROJ (pname, syar, stat)
KETW (key)
RSR RP (rname, pname)
RE (rname, ename)
PE (pname, ename)
PKL (pname, key)

図3.2 図3.1のLCS

表3.1 モデル要素の対応

LIS要素 ＼ E-Rモデル	値集合	属性	事象集合	関係性集合
リレーショナルモデル	定義域	属性	リレーション	リレーション間の関係性（外部キー等）
DBTGモデル	項目のオカーランス	項目名	レコード型	セット型 リンクレコード型
LISモデル	フィールドのオカーランス	フィールド名	セグメント型	階層パス

4. 問合せ変換（QT）

QTは同種化の逆過程であり、同種化で除去されたHIを用いて、LCSを参照する局所問合せ（LQ）からそのサイトで実行可能なDML群を生成する過程をいう。

4.1 仮定

QTを論じる上で、次のような仮定を設ける。

1) LQは、QUELによって記述されている。
2) LQは、aggregate関数を含まない。問合せ内のaggregate関数は、独立な問合せとして処理され、その結果の定数によって置き換えることができる。
3) 検索機能についてのみ検討する。
4) LQ内の結合（join）はLCS内のRSRに沿って書かれる。これ以外の結合は無意味とする。
5) 応答時間は、アクセスされるオカーランス数に比例する。
6) 目標DMLとして［DATEC77］に基づいたDBTGタイプのものを考える。このタイプを検討するのは、これが現存DBのDBMSとして今後最も有力と考えられるからである。
7) LQの条件部は積正規形である。

4.2 QTの構造

QTは図4.1に示すように、構造変換（ST），最適化（OPT），DML生成（DMLG）の3つの主要なモジュールからなっている。STはLQ内のLCS要素をDBTG要素に置き換える。この結果はDBTG問合せグラフ（DQG）と呼ばれるグラフにより表もされる。OPTは、DQGの全ノードとリンクとを訪ずれるアクセスパスの中で最適なものを見つけ、これを目標DBMSのアクセス単位に分割する。DMLGはこのアクセス単位を順に受けとり、目標DMLを生成する。

これらのモジュールが必要とする情報はHIとして各サイトがリレーショナル形式で管理している。

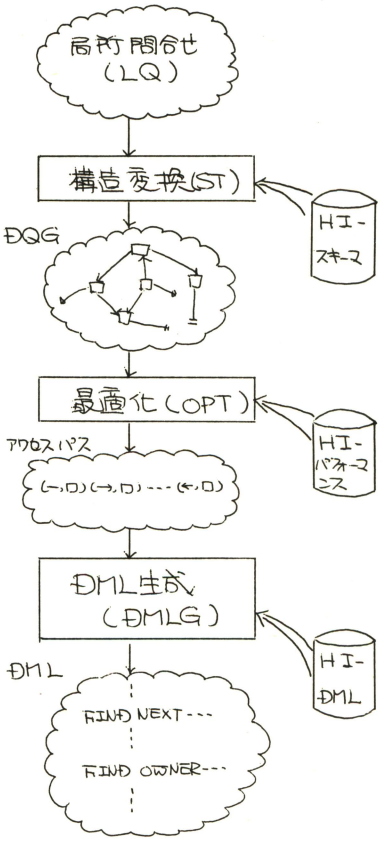

図4.1 QTの概要

4.3 例

3つのモジュールについて詳論する前に、本論文で用いている例について述べる。3章で示した例（図3.2）に対して次の問合せを考えよう："DBを1975年以降研究しているプロジェクトの代表者の部下が、同じプロジェクトに属している時、そのプロジェクト名と部下（ENG）の名前を求めよ"。これはQUELによって付記2のように記述できる。

5. 構造変換（Structure Transform or ST）

同種化は、LES要素を表3.1に基づいて1対1に対応させることによってLCSを生成する過程であるといった。STはこの逆過程であり、HIを用いて、　LQの参照するLCS要素を、対応するLES要素に置き換えることをまず考えねばならない。

5.1　リレーショナル問合せグラフ（RQG）

LQをグラフで表わすと、問合せの意味を理解する上で便利である[WONG77]。付記2のLQを表わすグラフを図5.1に示す。これは、ノードと3種のリンクとから成っている。ノードは組変数を、ノード間のリンクは結合(join)を表わしている。他の2種の終端リンク（→と⟼）は、各々制限（restriction）と結果属性とを表わしている。

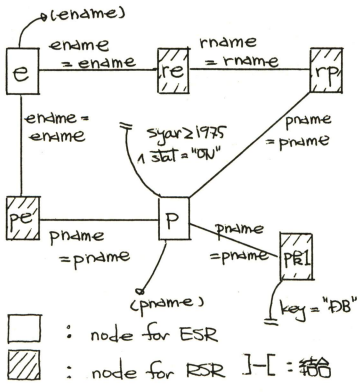

図5.1　RQG

ここで、このグラフが結合及び制限から成る論理式の積のみを表わしていることを注意しておく。即ち、（x.a=y.a ∨ y.b=z.c）といった和は表わせない。このためLQは、各サイトに発せられる以前にQDにより積正規化されているものとする。ただし、条件部の各節の論理式（結合又は制限）は全て同一の変数を参照していなければならない。即ち、各節はグラフのリンクに対応させられる。このようなLQをMCNF（modified conjunctive normal form）であるといい、これのグラフ表現をリレーショナル問合せグラフ（RQG）と呼ぶ。

5.2 隠れ構造（Hidden Structure or HS）

RQGを用いてLQの構造をDBTG構造へ変換しよう。まず、RQG内のLCS要素をDBTG要素へ置換する。例えば、ノードeはレコード型ENGを、rpはセット型RPを表わすことになる。この様にして、図5.2に示すグラフが得られる。箱はレコード型を、長円はセット型を示している。

ノードrを考えてみよう。rはレコード型REPRを表わしているが、LQ内には現われてこない。これは、LCS内のRSR（即ちREとRP）が、rの主キー属性rnameを持っているために、rnameのみを参照するLQはrではなくreとrpとを参照できるからである。この様に、LCSには存在するがLQ内には現われないレコード型をEHS（explicitly HS）と呼ぶ。

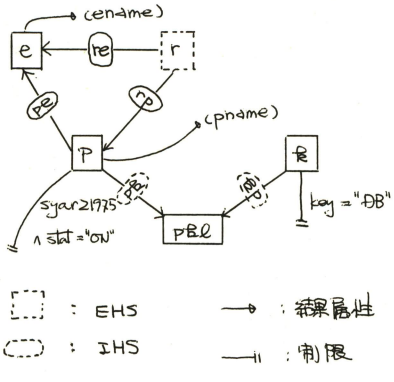

図5.2　HS

次にノードpBLとBPとを考えよう。両者は各々セット型PKとKPとを表わしている。図3.2のLCSから分かる様に、リンクレコード型PKLに関するこれらのセット型の情報はLCS上には存在しない。このように、LCS上にもLQ内にも存在しないセット型をIHS（implicitly HS）と呼ぶ。

EHSとIHSとを合わせて隠れ構造（hidden structure or HS）と呼ぶ。

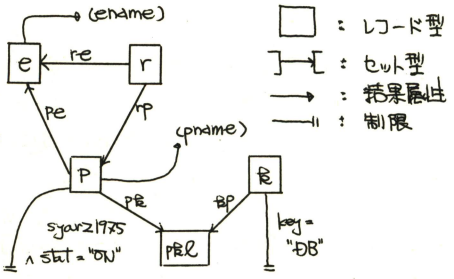

図 5.3 DQG

5.3 DBTG問合せグラフ (DQG)

EHSとEHSとは、異種性情報(HI)を用いて明らかにされる。この結果は、レコード型をノードとし、セット型をリンクとするグラフになる。我々は、これをDBTG問合せグラフ(DQG)と呼ぶ。

DQGは、LQの問合せの意味を、DBTG要素(レコード型、セット型)によって非手続的に表わしたものであり、STの出力となる。

6. 最適化 (OPT)

次の問題は、LQの非手続的DBTG表現(DQG)から手続を生成することである。これはDQGの全ノードとリンクとをたどるパスの中から最適なものを見つけることである。

6.1 目的関数

最適化の目的関数として、1)中間結果を最少とし、2)アクセスされるオカーランス数を最少とすることの2つを設定する。DBTGモデルは、リレーショナルモデルと異なり、DMLで生成された結果は、再度DBTGモデルとしてアクセスできない。即ち不完全である。従って、アクセスパスの途中で中間結果が生成されるならば、DBTGモデルとは独立なものとして別の処理系をつくる必要が生じてしまい、アクセスの一様性の点から望ましくない。このため第1の目標を達成することが、スキーマレベルでは最も重要となる。

2)は、より高速なアクセスを得るために必要となる。これは4章で述べた仮定5)に基づいている。

6.2 アクセスされるオカーランス数の見積もり

アクセスされるオカーランス数をOCAと記す。本節では、アクセスパスの生成に必要となるOCAの見積もりについて考える。ここで、n_iをDQGのi番目のノード、a_{ij}をn_iのj番目の属性、c_iをn_iの全オカーランス数、s_{ij}をa_{ij}の選択度(selectivity)とする。これらを用いると制限"$a_{ij}=v$"を満足するn_iのオカーランス数の平均値は$c_i \cdot s_{ij}$となる。又、s_iをn_iに関する等価制限式の選択度、s_i'をこのうちCALC項目を参照するものの選択度、s_i''をCALC以外のものの選択度とする。ここで$s_i = s_i' \cdot s_i''$であり、n_iがCALC等価制限式を持たなければ$s_i' = 1$、持てば$0 \leq s_i' \leq 1$である。

ノードn_iがアクセスパスの開始ノードとすると、OCA_iはn_iのアクセスモードに依存することになる。即ち$OCA_i = s_i' \cdot c_i$である。

ノードn_iがセット型s_{ji}を経由でn_jからアクセスされる場合を考えてみよう。ここでslj_iは、n_jの1オカーランスとSjiを介してリンクされているn_iのオカーランスの平均数とする。n_iがn_jの親であれば、$slj_i = 1$、逆であれば$slj_i \geq 1$である。これを用いると、$OCA_i = OCA_j \cdot slj_i$となる。

次に、任意のノード$(n_1, \cdots, n_k)$が線形にこの順にリンクされている場合を考えてみる。n_1を開始ノードとした時、このリンク内でアクセスされるオカーランス数(OCA)は、次の様になる。

$$OCA = s_1' \cdot c_1 (1 + s_1'' \cdot sl_{12} (1 + s_2 \cdot sl_{23} (1 + \cdots (1 + s_{n-2} \cdot sl_{n-2,n-1} (1 + s_{n-1} \cdot sl_{n-1,n})) \cdots)))$$

6.3 アクセス木（AT）

次に、DDGからアクセパスを生成する過程について論じる。アクセスパスは、DDG内のリンクとそれに結合されたノードとの対を訪ずれる順序に対応している。このパスは、一般にDDG内の全てのリンクをユニークに持ち、全てのノードを、あるものは冗長に持つ木として表わすことができる。これをアクセス木（AT）と呼ぶ。DDGがサイクルを持つ場合に、AT内に冗長ノードが現われる。この冗長ノードを合流ノード（confluent node or CN）と呼ぶ。CNはATにおいて重要な存在となる。DDGのあるノードrがAT内にれこの対応するノード（即ちCN）を$(t_1, \cdots, t_m)$を持つとしよう。$m \geq 2$であれば、$t_1, \cdots, t_m$をアクセスした各々の結果の共通部分（intersection）が求める解となる。中間結果の生成を最少とするためには、この操作を最少化せねばならない。

CNの生成はDDGからATの生成方法に大きく依存している。このため我々は横型（breadth-first）と縦型（depth-first）との2つの木生成について検討した。

6.4 横型AT生成（BFA）

この方法では、DDG内のあるノードrの隣接ノードを求め、OCAが小さい順にrの子ノードとしてリンクする。AT内のrと同レベルのノードに対して同じ事を全て行なう。ATにリンクされたDDGノードをOLDとマークする。既にOLDとなっているものが現われたなら、これはCNとなる。

図6.1は図5.3のDDGからeを開始ノードとすることによって作られたATである。eはCNである。この例からわかる様に、BFAでは異なった部分木内にCNが現われ得る。この時には、各々の部分木ごとに中間結果を生成し、これらの根ノード（p）で中間結果の結合が必要となってしまう。

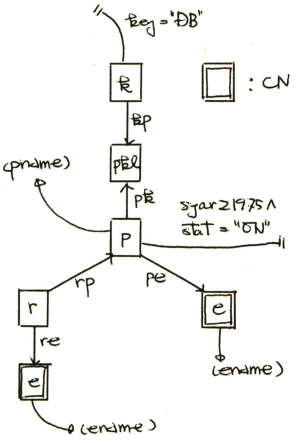

図6.1 BFAによるAT

6.5 縦型AT生成（DFA）

これに対して、同じくeを開始ノードした場合の縦型木生成によるATを図6.2に示す。この方法をDFAと呼び詳細を図6.3に記す。この例ではATはDDGのノードpについて2つのCNを持っている。便宜上AT内の始めに現われるものをp'、次のものをp''と名づける。前述したBFAとは異なり、ノードp''ではp'の主キー（pname）の値と現在の値とを比較し、等しい場合だけ答とすればよい。この様にDFAではCNが存在しても、BFAと異なり中間結果を生成しなくてよい。結果リレーションに、最初のCNがその主キーの値を出力しておき、次以降のCNは主キーの値を結果リレーション内のものと比較するだけでよい。同一であれば次のノードへアクセスに行き、そうでなければ

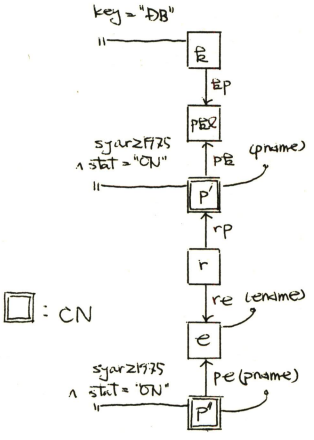

図6.2 DFAによるAT

アクセスを打切る。結果リレーションを用いてCNを処理出来、他に何らの中間結果を必要としない点はDFAの主要な利点である。

0. [初期化] STB(x)をDOGノードxの隣接ノードリストとする。STB(x)内には、ノードの対(x, g_1), ---, (x, g_n)が、OCA_{g_i}が小さい順に並べられている(即ち $OCA_{g_1} \leq OCA_{g_2} \leq \cdots \leq OCA_{g_n}$)。
DOG内の全てのノードはNEWとマークされている。 push down(Λ);
1. [開始ノードの決定] DOGの全てのノード内で最小のOCAを持ったものを開始ノードxとする。
2. push down(x); xがNEWとマークされていれば、OLDとマークする。
3. STB(x)が空ならば 10へ。
4. [STB(x)のサーチ] y ← Λ; i ← j ← 0;
5. [STB(x)内の次の対(x, g_i)をとり出す]
i ← i+1; iがnより大きければ 9へ。
STB(x)からi番目の対(x, g_i)を取り出す。
g_iがNEWで y ≠ Λならば、5へ。
6. g_iがNEWで y = Λならば、y ← g_i; [g_iは最初とNEWとマークされたノード]
8へ。
7. g_iがOLDならば、g_iをCONFLUENT(CN)とマークする。
8. j ← j+1; (x, g_i)をxのj番目の枝としてATへリンクする。
(x, g_i)をSTB(x)から消去する。 5へ。
9. [STB(x)内の全ての対はアクセスされた] y ≠ Λならば、x ← y として 2へ。
10. [popup] pop up(x);
xがΛならば stop; そうでなければ 3へ。

図6.3 DFA

6.6 ATからアクセスパスの生成

アクセスパスはATのノードを preorder で訪ねた場合の枝とノードその対のシーケンスによって表わされる。この対は、同時に目標DBMSのアクセス単位、e.g. DBTGの FIND NEXT ---, FIND OWNER ---, となっている。この枝とノードの対をアクセス単位(AU)と呼ぶ。

例えば、図6.2から得られるアクセスパスは((-, a)(up, pa)(pa, p*)(rp, r)(re, e)(pe, p*))となる。*のついた変数はCNを表わしている。

7 DMLの生成(DMLG)

最後のDMLGは、アクセスパスを受け取って、各アクセス単位(AU)ごとにDMLを含むCOBOLプログラムを生成し、全体として求めるプログラムを出力する。AUは、形式的に次のように記述できる。

```
AU ::= ( <unit-no> ( SET <set-type>) ( REC <record-type> (RSLT <result-attribute-list>)
                                        (CEQR <CALC-equi-restr>) (RSTR <restriction>) )
       (LITYPE <OWNER/MEMBER> <confluent-mode> < LEAF/START/INTERMEDIATE>)
       (ERTRN <unit-no> )
       (TRTRN <unit-no> )      )
```

<set-type>はATの枝に対応し、<record-type>は枝の下方についたノードに対応する。<record-type>が<set-type>の子である時NEXTノード、親である時OWNERノードと呼ぶ。ERTRNとTRTRNは<set-type>を介してリンクされた<record-type>の全てのオカーランスをアクセスした時の復帰先を示している。ERTRNは1つも満足するものがなかった時、TRTRNは少なくとも1つは満足するものがあった時の復帰ユニット番号である。AT内のノードrのERTRNは、rの親がNEXTノードならばそれであり、異なれば親のERTRN先となる。TRTRNはrが右手に兄弟ノードを持っていればそれであり、なければ親がNEXTノードならばこの親であり、異なれば親のTRTRN先となる。

AUは8つのクラスに分類される。先ずノードが葉、中間、開始、独立かによって4つに分けられる。最初の2つは更に、NEXTかOWNERかによって各々LAST-NEXT, LAST-OWNER, NEXT, OWNERに分けられる。後の2つはアクセスモードがCALCかどうかによって各々、CALC, FIRST, I-CALC, I-FIRSTに分けられる。各AUはこの8つのパターンについて照合が行なわれ、マッチすれば、それに対応するDMLが自動的に生成される。パターンとDMLとを付記3に示す。

この様にして図6.1から生成されたDMLを付記4に示す。

8. 結論

本論文では、QTについてQUEL問合せからDBTG DMLへの変換問題に焦点を当てて論じた。この中で、変換プロセスの詳細と必要情報とを明らかにした。構造変換の章ではLCSに基づいたLQとDBTGとの構造対応において明らかにされるべき隠れ構造を明確に示し、最適化の章では中間結果として結果リレーションを利用するアルゴリズム(DFA)の提案を行なった。中間結果の生成を最少とすることがスキーマレベルの最適化において最も重要であることから、DFAは有効である。更にATの分枝において、まず最少のOCAを持ったノードからアクセスすることによって、早い時期にアクセス空間を縮少することが可能である。この点は良い応答性能をもたらすと考える。

我々のQTは、実際のページアクセスを考慮していない。しかしページアクセスはDBMSの内部スキーマレベルの問題であり、そのインプリメンテーションに依存している。我々は中間結果の生成を最少とすることがスキーマレベルでの最適化の最重要項目であると確信している。従ってアクセスオカーランスの最少化も上述した方法以外は考慮しなかった。

DBTG以外の他のモデル、例えばIMSに対しても我々のQTは容易に対応できると考える。既存データモデルは、事象集合間の関係性集合、例えばDBTGのセット型IMSの階層パスによって特徴づけられ、これらはアクセスの単位でもある。この様な関係性集合の情報をHIに保持することによって、他のモデルへの拡張も可能である。

最後に我々のQTはインプリメンテーションの最終段階にはいっている。即ちBFAによるインプリメントを終り、DFAによるインプリメントを進めている。GCSからLCSへの問合せの分割(QH)についてもインプリメントを開始している。QT及びQHの完成後、これらを結合しJIPNETを用いて総合的な評価を行なうことを予定している。QTは当財団のM-160上にPL/Iを用いてインプリメントしている。

参照文献

[ADIBM 78] Adiba, M. and Euzet, C., "A Distributed Data Base System Using Logical Relational Machines," Proc. of the 4th VLDB, Berlin, Sept. 1978, pp.450-461.

[DATEC 77] Date, C. J., "An Introduction to Data Base System,"(2nd ed.) Addison-Wesley,June 1977.

[HELDG 75] Held, G., Stonebraker,M., and Wong, E., "INGRES - A Relational Data Base System," AFIPS Conf. Proc., May 1975, pp.409-416.

[ROTHJ 77] Rothnie, J, B. and Goodman, N., "An Overview of Preliminary Design of SDD-1: A System for Distributed Databases," Proc. of the 2nd Berkeley Workshop on Distributed Data and Computer Networks, May 1977, pp.39 - 57.

[STONM 77] Stonebraker, M. and Neuhold, E., "A Distributed Data Base Version of INGRES," ibid, pp.19 - 36.

[TAKIM 78] Takizawa, M., Hamanaka, E., and Ito,T., "Resource Integration and Data Sharing on Heterogeneous Resource Sharing System," Proc. of the ICCC'78, Kyoto, Sept. 1978, pp.253-258.

[TAKIM79a] Takizawa, M. and Hamanaka, E., "The Four-Schema Structure Concept as the Gross Architecture of Distributed Databases," to appear in JIP of IPSJ, 1979.

[TAKIM79b] Takizawa, M. and Hamanaka, E., "Query Translation in Distributed Databases," JIPDEC TR 79/04, Oct. 1979.

[TSICD 78] Tsichritzis, D. and Klug, A.(eds.), "The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Data Base Management Systems," Information Systems, Vol. 3, No. 3, 1978, pp.173-191.

[WONGE 77] Wong, E., "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases," Proc. of the 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977, pp. 217-235.

付記1　HI (Heterogeneity Information) スキーマ

HIは、ESR, RSR, ATTと呼ばれる3つのリレーションから成っている。ESRとRSRは、各々実線集合、関係性集合を表わすリレーションの情報を持ち、ATTはリレーションに属する属性についての情報を持っている。これらのスキーマを下記に示す。

ESR (entity-set-name, area-root-name, numb-of-keys, acc-mode, degree, width, size,
protection-flag, integrity-flag, cardinality)

RSR (relationship-set-name, s-es-name, d-es-name, acc-mode, rs-construct, degree, width,
size, s-set-name, d-set-name, protection-flag, integrity-flag, cardinality)

ATT (es-rs-name, es-rs-type, att-no, att-name, value-set, role-of-att, type, length, selectivity,
protection-flag, integrity-flag, cardinality)

付記2　LQの例

```
range of ( e,  p ) ( ENG, PROJ );
range of ( re, pe, rp, pkl ) ( RE, PE, RP, PKL );
retrieve into R ( p.pname, e.ename )
where pkl.key  = "DB"    and pkl.pname= p.pname  and p.pname = rp.pname and
      rp.rname =re.ename and re.ename = e.ename  and e.ename = pe.ename and
      pe.pname = p.pname and p.syar ≥ 1975       and p.stat  = "ON" ;
```

付記3　アクセスクラスとDML

```
FIRST       LFOUND ← false.
            FIND FIRST $01.   go to %02.
      %01.  FIND NEXT  $01.
      %02.  if endset = "yes",go to %06.
            GET $01.
            if not( $03 and $04 ), go to %01.
            call RESULT ( $05 ).
```

```
CALC        LFOUND ← false.
      %02.  call GNV (( $03 ), val, mode ).
            if mode="end",    go to %06.
            move val to $08 in $01.
      %01.  FIND DUPLICATE $01.
            if notfound="yes",go to %02.
            GET $01.
            if not( $03 and $04 ), go to %01.
            call RESULT ( $05 ).
```

```
NEXT    %01.  if %00 is true,   go to %02.
       *%00 is initially set false.
             if $02 is empty,  go to %06.
             %00 ← true.  %04 ← false.
                          go to %03.
        %02. %04 ← %04 ∨ LFOUND.
        %03. FIND NEXT $01 WITHIN $02.
             if endset="no",   go to %05.
             LFOUND ← %04.  %00 ← false.
             if %04 is false,  go to %06.
                          go to %07.
        %05. GET $01.
             if not( $03 and $04 ), go to %03.
             call RESULT ( $05 ).

OWNER   %01. FIND OWNER WITHIN $02.
             if notfound="no", go to %08.
        %02. LFOUND ← false.  go to %06.
        %03. GET $01.
             if not( $03 and $04 ), go to %02.
             LFOUND ← true.
             call RESULT ( $05 ).
```

```
LAST-   %01.  if %00 is true,   go to %04.
NEXT   *%00 is initially set false.
             if $02 is empty,  go to %06.
             %03 ← false.  %00 ← true.
        %04. FIND NEXT $01 WITHIN $02.
             if endset="yes",  go to %02.
             GET $01.
             if not( $03 and $04 ), go to %01.
             call RESULT ( $05 ).   go to %04.
        %02. LFOUND ← %03.  %00 ← false.
             if %03=false,     go to %06.
                               go to %07.

LAST-   %01. FIND OWNER WITHIN $02.
OWNER        if notfound="no", go to %03.
        %02. LFOUND ← false.  go to %06.
        %03. GET $01.
             if not ($03 and $04),go to %02.
             LFOUND ← true.
             call RESULT ($05).   go to %07.
```

(I-FIRST, I-CALC は省略されている)

ここでAUは次のようである。[7章を参照]

((<unit-no> (SET $02) (REC $01 (CEQR $03) (RSTR $04) (RSLT $05))
(UTYPE(a)(b)(c)) (ERTRN $06) (TRTRN $07))

%のついた数は1つのユニット内の内部変数である。特に%01はユニット番号の役目も持つ。

付記4 QTの例

```
RETRIEVE INTO RESULT1(P.PROJNAME,E.ENGINAME)
WHERE K.KEYWORD='DATABASE' AND P.PROJSTAT='ACTIVE'
AND P.PROJSYAR=1970
AND K.KEYWORD=PKL.KEYWORD AND PKL.PROJ-NO=P.PROJ-NO

AND P.PROJ-NO=RP.PROJ-NO AND RP.REPR-NO=R.REPR-NO
AND R.REPR-NO=RE.REPR-NO AND RE.ENGINAME=E.ENGINAME
AND E.ENGINAME=PE.ENGINAME AND PE.PROJ-NO=P.PROJ-NO;
GO;
```

```
      %---------- DBTG DML ----------

              MOVE FALSE TO LFOUND.
       L0102. CALL  GET_NEXT_VALUE ((KEYWORD = 'DATABASE' ),VAL,MODE).
              IF  MODE = 'END',  GO TO TERM.
              MOVE VAL TO KEYWORD IN KEYWORDS.
      % DUPLICATE ENTRY %
       L0101. FIND DUPLICATE KEYWORDS.
              IF  NOTFOUND = 'YES',  GO TO L0102.
              GET KEYWORDS.

       L0201. IF  L0200 = TRUE,  GO TO L0202.
              IF  KEYU-PROJ IS EMPTY,  GO TO L0101.
              MOVE TRUE  TO L0200.
              MOVE FALSE TO L0204.       GO TO L0203.
      %
       L0202. IF  L0204 = TRUE  OR  LFOUND = TRUE,  MOVE TRUE TO L0204
                                    ELSE     MOVE FALSE TO L0204.
       L0203. FIND NEXT PROJ-KEYU-LINK  WITHIN KEYU-PROJ.
              IF  ENDSET = 'NO',  GO TO L0205.
              MOVE L0204 TO LFOUND.       MOVE FALSE TO L0200.
              IF  L0204 = FALSE,  GO TO L0101.
                                  GO TO L0101.
       L0205.
              GET PROJ-KEYU-LINK.

      % OWNER ACCESS UNIT %
       L0301. FIND OWNER  WITHIN PROJ-KEYU.
              IF  NOTFOUND = 'NO',  GO TO L0303.
       L0302. MOVE FALSE TO LFOUND.
              GO TO L0201.
       L0303.
              GET PROJECT.
              IF  NOT ( PROJSYAR =  1970 AND PROJSTAT = 'ACTIVE' ),  GO
                TO L0302.
              MOVE TRUE TO LFOUND.
              CALL RESULT (PROJNAME ).

      % OWNER ACCESS UNIT %
       L0401. FIND OWNER  WITHIN REPR-PROJ.
              IF  NOTFOUND = 'NO',  GO TO L0403.
       L0402. MOVE FALSE TO LFOUND.
              GO TO L0201.
       L0403.
              GET REPRESENTATIVE.
              MOVE TRUE TO LFOUND.
```

```
       L0601. IF  L0600 = TRUE,  GO TO L0604.
              IF  REPR-ENGI IS EMPTY,  GO TO L0201.
              MOVE FALSE TO L0603.
              MOVE TRUE  TO L0600.
       L0604. FIND NEXT ENGINEER  WITHIN REPR-ENGI.
              IF  ENDSET = 'YES',  GO TO L0602.
              GET ENGINEER.

              MOVE TRUE TO L0603.

              CALL  RESULT (ENGINAME ).

              GO TO L0604.

       L0602. MOVE L0603 TO LFOUND.

              MOVE FALSE TO L0600.

              IF  L0603 = FALSE,  GO TO L0201.

                                  GO TO L0501.

       L0501. IF  L0500 = TRUE,  GO TO L0504.

              IF  PROJ-ENGI IS EMPTY,  GO TO L0201.

              MOVE FALSE TO L0503.

              MOVE TRUE  TO L0500.

       L0504. FIND NEXT ENGINEER  WITHIN PROJ-ENGI.

              IF  ENDSET = 'YES',  GO TO L0502.

              GET ENGINEER.

              MOVE TRUE TO L0503.

              CALL  RESULT (ENGINAME ).

              GO TO L0504.

       L0502. MOVE L0503 TO LFOUND.

              MOVE FALSE TO L0500.

              IF  L0503 = FALSE,  GO TO L0201.

                                  GO TO L0201.
```

```
RESULT (REPRESENTATIVE  ) <-- JOIN RESULT (REPRESENTATIVE  ) AND RE
SULT (E
NGINEER         ) ON ENGINEER         .ENGINAME
```