データベース・システム 82－5
（1991. 3. 11）

リンク指向 DBMS G-BASE における リンク機能の拡張とその応用

平岡 昭夫
(株)リコー ソフトウェア研究所

G-BASE はグラフデータモデルに基づく本格的 DBMS である。グラフデータモデルは関係データモデルに対し、レコード間の関連を簡潔に表現できるリンク機能を拡張した新しいデータモデルである。我々は、より複雑な構造のデータを表現できるように、リンクに関連についての属性情報を持たせる拡張を行った。本稿では、リンク機能によるレコード間の関連の表現方法を紹介するとともに、今回おこなった機能拡張を報告する。さらに、リンクを使った複数レコード間の検索について簡単な性能評価を行っている。また、応用例として階層構造を持つ部品表データベースを紹介する。

Extension and Application of the Link Facility in the Link Oriented DBMS G-BASE

Akio Hiraoka
Ricoh Software Research Center
1-1-17, Koishikawa, Bukyo-ku Tokyo 112, Japan

G-BASE is a DBMS based on the Graph Data Model. The Graph Data Model is a new data model which is an extension of the Relational Data Model. It allows the relationship between record occurrences to be represented intuitively via links. The link facility has been extended to allow for the description of relationship attributes. This paper explains how the link facility can be used to represent the relationship between records and reports on the newly expanded features. It contains a performance evaluation of retrievals using links versus retrievals without links. An example of an application (a Parts List Database with layered structure) is also provided.

1 はじめに

UNIX/WS[1]上のDBMSとしては、リレーショナルデータモデルをベースとしたDBMSが現在主流であり、標準データベース操作言語SQL [1]は普及段階に入っている。E.F.Codd [2]によって提案されたリレーショナルデータモデルは、明確な数学的基盤に基づいているうえに、判り易いことも手伝って、UNIX/WS上の主流データモデルとなっている。

ところが、ワークステーション上では、従来の事務処理にはなかったデータベースの応用分野が広がりつつある。たとえば、CADシステムや、画像処理システム、文書処理システム、マルチメディアシステムなどの核にDBMSを使用する研究が多数試みられている[3][4]。

このような分野に利用されるDBMSは多様で複雑なデータを扱わなければならない。たとえば、個々のデータが複雑な関連をもっていて、ある関連によって結びつけられるデータの集合が意味をもつことがある。機械設計の分野における階層構造データはこの代表例である。また、文書構造を表す場合などにも階層構造によるデータの管理は重要である。単純な表構造しかもたないリレーショナルデータモデルは、複雑なデータを表の組みわせで表現しなければならないため、ユーザに対して大きな負担を強いている。

最近ではこのような問題に対して新しいデータベースとして、オブジェクト指向データベース(OODB)や、演繹データベース(DDB)等が注目されている[5]。OODBについては、いくつかの実用的なシステムが開発されている一方で、基本概念に対する総意の一致が見られていないのが現状である[6][7]。また、データ制御に関する基本機能の実現方法についても課題が残されている。一方、DDBは高い推論能力を長所としながら、モデル化能力に問題点があると言われている[8]。

我々はこれらの問題に対して、レコード間の関連を直接表現できるリンク機能を提案してきた[9]。リンクは、**グラフデータモデル**[10]と呼ばれる新しいデータモデルの機能の一部である。G-BASE[2][11]はグラフデータモデルをベースに開発された本格的DBMSである。

本稿ではリンク機能に対して行った機能拡張を紹介し、リンク機能を階層データに応用した例について説明する。

[1]UNIXはAT&Tが開発しライセンスしています。

[2]G-BASEは(株)リコーが開発し、販売しているデータベース管理システムの名称です。

2章ではグラフデータモデルについて、リンク機能を中心に説明する。3章ではG-BASEの新しいバーションで拡張したリンクフィールド機能について、実現方法とともに紹介する。4章はリンクを使った検索の簡単な性能評価を行っている。5章はリンク機能の階層構造データに対する応用例として部品表管理システムを紹介している。

2 グラフデータモデル

リレーショナルデータベースの持つ簡潔さを保ちつつも、データ間の関連を直観的に表現できるように拡張されたモデルがグラフデータモデルである。

データベースの設計を行う際に、実世界のデータ構造を表現する概念モデルとして実体関連(E-R)モデル[12]が使われることが多い。リレーショナルモデルの場合この概念モデルから正規化という手順を踏んでより効果的なデータベース設計を行わなければならない。ところが、正規化を行うことで表は分割され、さらに表と表の関連を表すための中間表(関連表)などを用意することから、表の数が多くなり、操作時には表間の関連をつかみにくくなるという問題がある。

グラフデータモデルではこれらの問題を解決するためデータモデルの中にタプルとタプルの関連を直接表すことのできるリンク型というオブジェクトを導入した。このグラフデータモデルをベースに開発されたDBMSがG-BASEである。

グラフデータモデルではリレーショナルデータベースで云う表を**レコード型**と呼んでいる。そして、各行を**レコード**(または、レコードオカレンス)と呼ぶ。レコード型を構成する列を**フィールド**と呼ぶ。複数のレコード型、リンク型をまとめたものをデータベースと呼ぶ。システム中に作成できるデータベースの個数に制限はない。

G-BASEの場合データベースの構造(スキーマ)を表現するために、データ定義言語(DDL)を使用する。図1に簡単な人事データベースのDDL文による定義例を示す。**社員**と**組織**という2つのレコード型が定義されている。フィールドごとにその名前とデータ型が宣言されている。

ここで、社員が所属する組織のレコードとの関連をリンク型で表現するためには次のように宣言する。

```
real link 社員 ../所属/.. 組織;
```

```
database "/demo/jinji";

record 社員 (
    社員番号   integer(4),
    氏名       character(32),
    住所       variable character(256),
    生年月日   date,
    趣味       variable character
) key(社員番号);

record 組織 (
    組織名     character(64),
    売上高     integer(4),
    担当分野   variable character(256)
) key(組織名);
```

図 1: 人事データベースのスキーマ

図 2: 人事データベースのスキーマ図

この宣言文で**社員**レコード型と**組織**レコード型の間にリンク型**所属**[3] が定義され、**社員**と**組織**の間の関連を表現できるようになる。

G-BASE の中でリンク型はつねに ../<リンク名>/.. という記法で定義、参照される。リンク型は 2 つのレコード型間でのみ定義される。図 2 はこのデータ定義を E-R モデルで表現した図(スキーマダイアグラム) である。

G-BASE ではこのように宣言したスキーマファイルを DDL プロセッサにかけることで、データベースが作成される。

作成されたデータベースには必要なデータを格納しなければならない。データベースのデータに対する操作はデータベース操作言語(DML)によって表され、DML プロセッサによって実行される。レコード型にレコードを格納する場合は insert 文を使用する。

```
insert 社員 set 社員番号 = 230199,
               氏名 = "平岡",
               住所 = "東京都中野区",
               生年月日 = '1959/11/9',
               趣味 = "読書";
```

レコード型にレコードを格納しただけではリンクは接続されない。両端のレコードを指定し、connect 文を使って、リンクを接続する。

```
for 社員[名前 = "平岡"]..組織
    [名前 = "ソフトウェア事業部"]
    connect 社員 ../所属/.. 組織;
```

for 文は必要なレコードの集合を求めるための検索構文で、ここでは**社員**、**組織**それぞれのレコード型に対して [ ] 内の条件を満たすレコードの組が結果として得られる。connect 文は for 文で得られたレコードの組に対して実行されることになる。レコード型のレコードを削除する場合には delete 文を使用し、リンクを切断する場合には、disconnect 文を使用する。

G-BASE の DML ではレコードを検索する操作はパス集合を作ることによって実行される。パスとはレコード型と実リンク型を交互にたどってえられるレコードの集合で**パス式**によって表現される。

図 3 において始端レコード(1 ～ 4)からリンク(矢印)をたどって得られるレコードの組がパスである。パスのあつまりがパス集合である。1 つのレコードに複数のリンクが接続されている場合には、それぞれをたどったレコードの組が別のパスとして得られる。

社員と**組織**の関連**所属**リンク型をたどって得られるある組織に属する社員(と指定した組織)のパスは次のようなパス式によって表現される。

```
社員 ../所属/.. 組織[組織名 = "～"]
```

前述の for 文はこのパス式で作られるパス集合の各パスに対する操作を指定する構文である。たとえば、パス式上のフィールド値を得る場合には print 文を for + パス式の後に指定すればよい。

```
for 社員 ../所属/.. 組織[組織名 = "～"]
    print 社員.氏名, 社員.住所;
```

[3] リンク型にはそのデータの持ち方で『実リンク(real link)』と『仮想リンク(virtual link)』の 2 とおりがあるがここでは、同じものとして説明する。

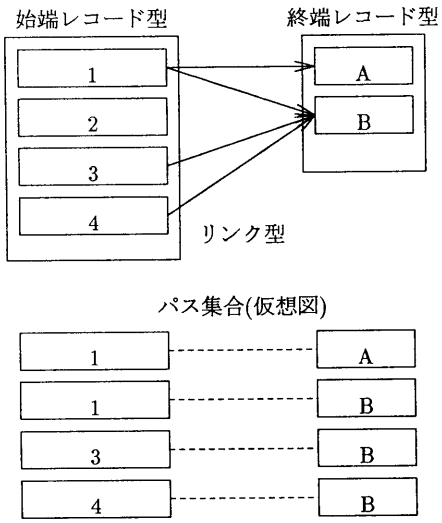

図 3: パスファイルの概念図

リンク型の両端にあるレコード型をそれぞれ始端レコード型、終端レコード型と呼ぶ。パス式中でリンクは左から右にたどられる。リンク型は双方向にたどることができるが、リンクの定義に対して逆方向にたどる場合には **組織** ../inverse **所属**/.. **社員** のように inverse を指定する。逆方向のリンクに対して名前を指定することもできる。

パス式のもっとも簡単な形式はレコード型１つである。複数のレコード型、リンク型をたどるにはパス式中にレコード型、リンク型を繰返し指定すればよい。

社員 ../所属/.. 組織 ../場所/.. 事業所

G-BASE ではリンクを使用しなくても、レコード型それぞれに対してリレーショナルデータモデルに対する操作と同じ集合演算(union, join, difference 等) が用意されている。

図 2 の例では**社員**と**組織**という**多対１**の簡単な関連を表しているため、リレーショナルデータモデルでも簡単に表現できる。しかし、社員が複数の組織に兼務している場合、一般にリレーショナルデータモデルでは中間表を用意しなければならなくなる。このような場合でも、グラフデータモデルではリンク機能を使って**多対多**の関連を簡潔に表すことができる。

以上のようにデータ間の関連を表すリンク機能によって、G-BASE は E-R モデルをほとんどそのまま表現でき、リレーショナルデータモデルで問題となる正規化の手続を大幅に削減している。

ところが、このリンク機能にはいくつかの問題点があった。最も大きな問題点は、リンクつまりはレコード間の関連に対して情報を持たせることができない点であった。現在のリンクは単にレコード間をポインタのようにつないでいるだけで、その関係に対して情報記述機能を提供していない。したがって、ある条件に合う関係を持つレコードの組を検索するような操作を記述できなかった。たとえば、先の例で**社員**の兼務の割合をリンク情報に付加したい場合、そしてある社員について兼務の割合が 0.5 を割っている所属を出力したい場合など、従来のリンク機能では対応できなかった。

次章ではこの問題点を解決するため、今回我々が行ったリンクに対する拡張機能を説明する。

3 拡張されたリンク機能

我々は、G-BASE のリンク機能をより完全なものにするため、リンク(関連)に対して情報を持たせることを可能にした。リンクに持たせた情報のことを**リンクフィールド**と呼んでいる。

例えば、**業者**と**製品**の間に**納入**という関係がある場合を考える。

```
database "/demo/納品管理";

record 業者 (
    社名        character(64),
    住所        variable character(256)
);

record 製品 (
    製品番号    integer(4),
    製品名      character(128)
);
```

ある業者がある製品を納入しているという関連は**業者** ../**納入**/.. **製品**というリンク型で表される。しかし、業者がその製品を単価いくらで、どれだけ納入しているかを表すためには、**納入**というリンクに対して、単価や個数といった情報を持たさなければならない。我々はこれらの情報を**リンクフィールド**として定義できるようにリンク機能を拡張した。

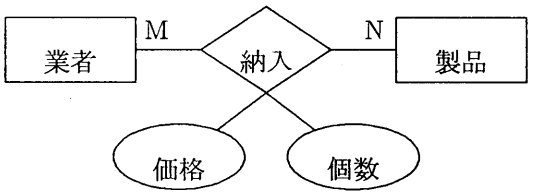

図 4: 納品管理データベースのスキーマ図

リンクフィールドを含むリンクの宣言の例を示す。

```
real link 業者 ../納入/.. 製品 (
    単価        integer(4),
    個数        integer(2),
    納期        data
);
```

リンクを接続する connect 文もリンクフィールドに値を代入できるように拡張された。

```
connect 業者 ../納入/.. 製品
    set 単価 = 198000, 個数 = 20;
```

またパス式の中にリンクフィールドの値に対する条件を指定できるように拡張した。レコード型のフィールドとリンク型のフィールドを区別するため、フィールドを参照する時にはレコード型名、リンク型名を指定する。

```
for 業者 ../納入/.. 製品
    [製品.製品名 = "ネジA"
     and 納入.個数 >= 10]
    print 業者.社名, 業者.住所,
          納入.個数;
```

この例[4]ではネジAを１０個以上納入している業者を検索し、その社名、住所と納入個数を出力している。

今回のリンクに対する拡張により、G-BASE は E-R モデルで表現された概念スキーマを、そのまま表現できるようになった。図４にレコード型**業者**,**製品**とリンク型**納入**についてのスキーマ図を示す(レコード型のフィールドは省略している)。

一般にリレーショナルデータベースでは、納入という関連をユーザが作成する表で管理しなければならない。業者、製品リレーションに個々のタプルを識別するフィールド(結合列)を用意し、そのタプル間の組をこの表に格納する。格納されるタプルの識別子を持つ業者、製品が実際に存在することもユーザの責任で調べなければならない[5]。

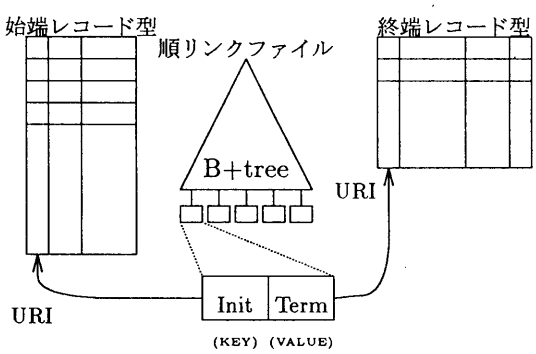

図 5: リンクファイルの構造

例のような単純なスキーマならば簡単かもしれないが、実際に使用される複雑なスキーマのなかで、関連するテーブルを管理していくのはユーザにとって大きな負担である。さらに、**製品 → 業者**、**業者 → 製品**の検索を行う場合、3 つの表の結合オペレーションを行わなければならないため、データ量が多くなった場合、性能が著しく低下することが予想される。

G-BASE でリンク型を定義したした場合、2 つの**リンクファイル**が作られる。それぞれを**順リンクファイル**、**逆リンクファイル** と呼ぶ。各ファイルは B+tree(B+木)で構成され、キーとしてリンクが接続されている始端レコードの**レコード識別子**(URI: Unique Record Identifier) が格納される。サーチの結果得られる値は終端レコードの URI である。これらのファイルを検索することで、リンクを順方向、または逆方向に高速にたどることができるようになっている(図 5 参照)。

新たに追加した機能であるリンクフィールドは**リンクレコードファイル**に格納される。2 つのリンクファイルはこのリンクレコードファイルに対する索引ファイルの関係になっている。これらのファイルはすべてリンクの接続/切断操作にしたがって自動的に更新される。

以上のような物理的な構造はユーザからは見えない。

[4] 条件式 [～] をパス式の右端に集めている場合、どの条件から順番に適用していくかを DBMS の最適化に任せることになる。

[5] アトリビュート間の参照一貫性のインテグリティ指定をできるシステムもある。

ユーザにとって、リンクはあくまで、connect 文で接続し、disconnect 文で切断するだけの論理的な構成要素に見える。

リンクフィールドの追加に伴い、リンクに対する update 操作が追加された。また、従来同じレコードオカレンスの組の間には１本のリンクしか接続できなかったが、今度の機能拡張で可能になった。たとえば、同じ業者が同じ製品を異なる値段で納入するという情報も表現できるようになった。リンクの順方向/逆方向それぞれに異なる値を持たせることも検討したが、あまりニーズがなかったので今回の拡張からは除かれた。

4 リンクによる検索の性能評価

次に、G-BASE においてリンクを使用しなかった場合と、使用した場合のの性能の違いについて考察する。

G-BASE ではレコードは**レコードファイル**に格納される。このファイルは**順次アクセスファイル**である。すべてのレコードは一意に識別されるための URI を持つ。URI には物理ファイル内の位置に基づく値が設定されるため、URI からレコード値を得る場合はファイルに対する直接アクセスが可能である。

検索が頻繁に行われるフィールドについては**索引**を設定する。索引は B+木ファイルに格納される。この B+木から検索対象のフィールド値をキーとしレコードの URI を値として得ることができる。あるフィールド値を持つレコードを検索する場合には、索引ファイルをフィールド値でサーチした後に、得られた URI でレコードファイルを読む。

前章で述べたようにリンクファイルも始端レコードの URI をキーとし、終端レコードの URI を値として持つ B+木である。

G-BASE のカーネルは UNIX ファイルに対して固定サイズのページ単位にアクセスする[6]。また、レコードファイル、索引ファイル、リンクファイルの各タプルはページをまたがっては存在しない。以下では、簡単のためファイルから読み出さなければならないページの数を検索の時間と考えている。

B+木を検索するために読まなければならないページ数は B+木の高さに等しい。B+木の高さは、タプル数が N 個、各ノードに m 個の子ノードがある場合 $\log\lceil m/2\rceil N$ を越えない。ここで B+木のノードサイズを P とし、キー(+ 下位ノードへのポインタ)の長さを K とすると、m = P/K である。したがって、ノードサイズ P を固定とすると、B+木の高さはキーの長さ K、レコード数 N をパラメータとする次の式で表される。

$$F(K,N) = \log\lceil\frac{P}{2K}\rceil N + 1(P:\text{固定})$$

はじめに、あるレコード型 A からリンク型 L をたどってレコード型 B の値を得る場合を考える。レコード型 A, リンク型 L のタプル数, キーフィールドのサイズを次のように仮定する。

	タプル数	キーサイズ
レコード型 A	Na	Ka
リンク型 L	Nl	Kl

検索はすべて索引ファイルに対して行い、検索結果はつねに１個になるような検索のみを対象にする。レコードファイルは値を参照する場合にのみ URI から直接参照される。

ある始端レコード１つから１本のリンクをたどり、１つ終端レコードを得るためには、２個の B 木と、１つのレコードファイルをサーチすればよい。

1. レコード型 A の URI を得るため、索引ファイルをサーチする⇒ F(Ka, Na)
2. 始端レコードの URI から終端レコードの URI を得るため、リンクファイルをサーチする⇒ F(Kl, Nl)
3. レコード型 B の URI から、実際のレコード値を得る⇒ 1

結果として、リンクを使った場合には F(Ka, Na) + F(Kl, Nl) + 1 のコストでレコード間の関連が得られる。

一方リンクを使用せず、ユーザが指定した結合フィールド(結合列)によって中間表を検索する場合を考える。ユーザが指定した結合列はレコードの位置を表していわけではないので、索引ファイルをサーチしたあと、結合列の値を得るためレコードファイルを読まなければならない。また、中間表から得られた終端レコードの結合列値から実際のレコード値を得るために B+木をサーチしなければならない[7]。つまり、始端レコードか

[6] システム全体では、ページのサイズは 1KB 境界の可変長であるが、ファイル単位には固定長でバッファリングを行っている。

[7] 実際には中間表に対して索引ファイルが用意され、中間表に対する検索はこの索引ファイルに対して行われる。したがって、始端レコードの結合列から終端レコードの結合列を得るためにはレコードファイルを一度読まなければならないが、ここでは省略している。

らユーザが作成した中間表を介して終端レコードを得るためには、B+木を 3 回、レコードファイルを 2 回サーチしなければならない。

中間表に格納されるタプルの数はリンクファイルに格納されるタプルの数と同じ(Nl)であると考える。また、キーのサイズもほぼ同じ(Kl)と考えてよい。レコード型 B の個数, 結合列値からレコードの URI を得るための索引ファイルのキーサイズを次のように仮定する。

	タプル数	キーサイズ
中間表	Nl	Kl
レコード型 B	Nb	Kb

1. レコード型 A の URI を得るため、索引ファイルをサーチする⇒ F(Ka, Na)
2. URI から始端レコードの結合列を得るため、レコード型 A のレコードファイルを読む。⇒ 1
3. 始端レコードの結合列から終端レコードの結合列を得るため、中間表の索引ファイルをサーチする⇒ F(Kl, Nl)
4. 結合列の値から終端レコードの URI を得るためレコード型 B の索引ファイルをサーチする。⇒ F(Kb, Nb)
5. レコード型 B の URI から、実際のレコード値を得る⇒ 1

結果として、G-BASE でリンクを使わなかった場合、F(Ka, Na) + 1 + F(Kl, Nl) + F(Kb, Nb) + 1 のコストがかかり、使った場合に対して F(Kb, Nb) + 1 だけ余分なページを読み出す必要がある。つまり、結合を行うための結合列の値から実際の URI を得るための時間がそのまま余分なコストになるわけである。レコード型 B の個数 Nb が大きい場合にはこの差は非常に大きくなる。リンクを使用する場合、終端レコード型 B のレコード数に性能は依存しない。

この例は 1 個の始端レコードから 1 本のリンクをたどり、1 個の終端レコードを得るという非常に単純な検索である。また検索にすべてインデックスが使用できることを前提としている。始端レコードが複数個あったり、インデックスのないフィールドでの join 操作を行わなければならない場合、リンクを使った場合と、使わない場合の性能差はさらに大きくなると考えられる。

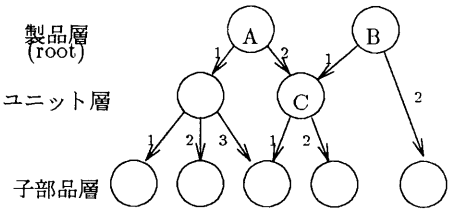

図 6: 部品表の木構造

5 階層型データに対するリンク機能の応用

我々は、G-BASE のリンク機能の有効な応用例として階層型データの取り扱いを考えている。以下では部品表管理を例にリンクを使って階層型データを表現し、操作する方法を考察した。

機械設計などの現場では、部品を管理するために部品表を使用している。部品表の構造の特徴は、親部品とその子部品からなる木構造である。木構造の葉にあたる部分に最小単位の部品情報が格納され、中間ノードは複数の部品から構成されるユニットである。いくつかのユニットの組みあわせによって製品が構成される。

部品表は親部品を起点とし最小部品単位にいたる木構造である。ところが、下位の部品またはユニットは複数の親ノードを持っている共通部品であることも許される。たとえば、C というユニットが A, B 両方の構成部品であることが許されなければならない。したがって、部品表木構造の上位ノード対下位ノードの関連は多対多の関連となっている(図 6 参照)。

部品表に対する代表的な検索操作は『ある部品の子部品すべてを得る(部品展開)』と『指定された部品を使っているすべての親部品を検索する』であり、部品表の木構造を下位方向、上位方向にトラバースする(たどる)操作が基本となる。

G-BASE ではこのような部品の階層構造を表すために、リンク機能を使用する。上位部品と下位部品の関連をリンクを接続することで表現する。実際の応用を考える場合、1 つの親部品に対する子部品群は単なる集合ではなく、親部品から見た時に個々を識別するべき情報が必要となる。また、同じ親部品を持つ子部品間での順序づけも必要な機能である。このような情報は子部品が持つべき情報ではなく、親部品との相対的関連に基づく属

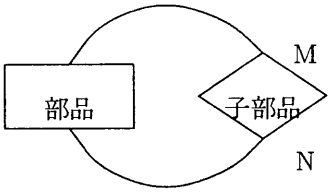

図 7: 部品管理 DB スキーマ図

性と考えられる。今回の拡張により、親部品-子部品間の関連情報をリンクフィールドで表現することが可能になった。

単純な部品のスキーマ構造を例としてあげる。最小単位の部品と中間ユニットを別のレコード型で表現する方が現実的ではあるが、ここでは簡単のためすべての部品(木構造のノード)を同じレコード型**部品**で表現している。

```
record 部品 (
    部品名     variable character(100),
    付属情報   character(150)
) key(部品名);

real link 部品 ../子部品/.. 部品 {
    番号       integer(4),
    色         character(16),
    倍率       double,
    位置[2]    double
};
```

部品間の親子関係を表現するためにリンク型**子部品**を定義した(図 7 参照、フィールドは省略)。このリンク型は同じレコード型に属するレコード間の関連を示している自己参照型リンクである。各リンクは親部品から見た子部品の番号と、変倍率、相対位置(X 座標,Y 座標)、色情報(カラー)をリンクフィールドとして持つ。同じ部品レコードも異なる番号、相対位置で別の親部品から参照される場合がある。

このような簡単なスキーマによって、先に述べた木構造を表現することができる。『親部品を 1 つ指定して子部品を検索する』検索操作は、リンク機能を使って次のように表現できる。

```
alias 親   部品;
alias 子   部品;

for 親[<条件式>] ../子部品/.. 子
    order 子部品.番号
    print 子.部品名;
```

レコード異称(alias)は同じレコード型の実体に対して、異なるスコープを与えるための宣言である。上記の例では< 条件式 >を満たす部品を 1 つ検索し、その部品を親部品として持つ子部品をリンクフィールド**番号**の順に出力する。

複数のレベルの親子関係を表現するには for 文を入れ子形式にすることで、実現できる[8]。

```
alias p0 部品;
alias p1 部品;
alias p2 部品;

target  t0 p0;
target  t1 p1;
target  t2 p2;

find t0 = p0[<条件式>];
for t0 {
    print Name, Id;
    find t1 = p0 ../子部品/.. p1
              order 子部品.番号;
    for t1 {
        print "    ", Name, Id;
        find t2 = p1 ../子部品/.. p2
                  order 子部品.番号;
        for t2 {
            ..................
        };
    };
};
```

上の検索では、基本的に親部品に対してすぐ下の子部品しか検索することができない。ある親が指定されたときにその子供をすべて再帰的に得るためには**ターゲット**を使用する。ターゲットとはパス式によって検索されたレコード群を一時的に保持するための表である。この表の実体はレコード型であり、**ターゲット異称**を宣

[8]指定レベルの子部品を得るためには応用プログラム側で再帰呼びだしを行うことで、子部品集合を順に得ることができる。

言することで同じレコード型上に異なるスコープを作ることができる。

```
var cnt;
alias 親  部品;
alias 子  部品;
target  tp  親;
alias   tc  target tp 子;

find tc = 子[<条件式>];
while cnt != 0
{
    set cnt = count(tc);
    find tc = 子[path tc or
            tp ../子部品/.. 子
            [子部品.色 = "赤"]];
}
```

上の検索文を実行することで、<条件式>に適合した親部品に対する子部品のなかで**色**が赤に指定されている子部品の集合を再帰的に得る。while 文の終了条件は、条件を満たすレコードをターゲットに追加してもターゲット上のパスの数が増えないことである。これは、現在ターゲット上にあるレコードに対して条件を満たす子部品が得られないことを意味する。

このように、G-BASE ではリンク機能を用いることで階層構造データを簡潔に表現できる。前章で述べたように、リンク機能を利用する場合、中間表を使った結合操作に比べて検索速度も速い。

以前のバージョンではリンクに対する属性を持たせることができなかったため、親部品に対する子部品の番号(または、順序)を指定することができなかった。これは、現実的な応用を考える場合大きな問題点であった。我々は、3 章で述べたリンクフィールド機能を拡張することによってこの問題を解決した。ここでは、階層構造のデータとして部品表を例としてあげたが、ほかにも文書の構造を表現する応用も研究されている。

6 今後の課題

前章で部品表のような階層構造を持つデータに対するリンク機能の応用例をあげた。構造を持つデータに対する表現能力はリンクフィールドを導入することでかなり高まった。しかし、パス式によりパス集合を作るというデータ操作の基本方式には、いくつかの問題点がある。

その一つは、リンクで接続されたレコード集合の**閉包**をとるという操作(リカーシブ・クロージャ)である。現在の版でもターゲットのもつスコープ機能を使うことで階層構造データのパス集合を作ることは可能であるが、出来たパス集合内で、パスの階層が表現出来ないという問題がある。そこで、われわれは、閉包を作るための新しいパス集合**クロージャパス**を追加したいと考えている。

クロージャパスとはレコード型１つ、自己参照リンク型１つを再帰的にたどって得られる閉包パス集合のことである。現在のパス集合は単純な表形式の一時ファイルに格納されるが、クロージャパスを実現するためには、ファイル構造を新しくしなければならない。さらに、複数のレコード型、リンク型にまたがる集合の表現方法についても検討していかなければならない。

リンクによって複雑なデータ構造を表現し操作する方式とは別に、データ自身に対して ADT (Abstract Data Type) 機能を導入していくことも検討されている。今後は、リンクの操作面での拡張と、ADT 機能の導入を目標に研究を続けていくつもりである。

7 まとめ

本稿では G-BASE のリンク機能とリンクフィールドの拡張について、その応用例を中心に紹介した。また、リンクを使った場合の性能についても考察した。性能については非常に限られた範囲でのモデルを対象にしているので、実測値をもとにさらに検証する必要がある。

G-BASE のリンク機能はリンクフィールド機能を追加したことによって、複雑なデータを簡潔に表現できるようになった。そして、より現実的な応用分野で利用できるようになったと考えている。今後は実際に様々な分野で応用し、その問題点を検討していくつもりである。

参考文献

[1] 国際規格: "Database Language SQL with integrity enhancement", ISO/IEC 9075-1989

[2] Codd,E.F.: "A relational model of data for large shared data banks", Commun. ACM, Jun 1970

[3] 飯沢,白田: "マルチメディア・オブジェクト指向によるデータベースの構築とその例",技研情報センターセミナー資料, Feb 1990

[4] 佐藤: "データベース機能を持つマニュアル作成支援システム", 情報処理学会データベースシステム研究会技術報告 90-DBS-78, 1990

[5] 小林: "古典データベースから演繹データベースへ",情報処理,V31, No.2,pp.189 ～ 197

[6] Atkinson,M. et al.: "The Object-Oriented Database System Manifesto", Proc.of the 1st International Conference on Deductive and Object-Oriented Databases (DOOD'89), pp40-57,Dec. 1989

[7] The Commitee for Advanced DBMS Function: "Third-Generation Data Base System Manifesto", Memorandum No. UCB/ERL M90/28, April 1990

[8] 横田,西尾: "演繹・オブジェクト指向データベース",情報処理,V31, No.2,pp.234 ～ 243

[9] 長: "グラフデータモデルを実現した G-BASE",情報処理学会第 35 回全国大会 7Cc-4, 1987

[10] Kunii,H.S.: "Graph Data Model and its Data Language", Springer-Verlag

[11] (株)リコー編:"G-BASE システムマニュアル", G-BASE マニュアル

[12] Chen,P.P.S.: "The Entity-Relationship Model - toward a unified view of data", ACM Trans. Database Syst., Mar 1976, 1(1)