関数型データベースにおける階層型検索データの生成

永江尚義[†] 有澤博[†]

本稿では関数型データモデルに対する新しい検索手法を提案する．関数型データモデルでは，現実世界のデータの意味を完全かつ特定の物理構造に制約されない平坦な形式でデータベースへ蓄積するために，単純なデータの二項関係だけを用いて情報をモデリングする．しかし，このような関数型データベースでは膨大なデータが複雑に絡み合うような構造によって表現されるデータ（例えば，文書や図形などの階層構造を持つデータ）をそのままの形でユーザに対して提供することはできない．なぜなら，構造を持ったデータもデータベースへ蓄積する際に単純な二項関係に細かく分解されるため，もともとのデータが持っていた階層などの構造情報がデータベース中ではさまざまな二項関係の中に埋没され，明確に区別されなくなってしまうからである．本稿ではデータベース中の二項関係のデータの集合から複合オブジェクトを生成するための構造化オペレータを定義する．このオペレータを用いることにより，データベース中の二項関係には本質的な上下関係がないために，1つのデータベースから容易にさまざまな階層構造を持つ複合オブジェクトを生成することができる．

The Construction of Hierarchical Data Structure from Functional Databases

HISAYOSHI NAGAE[†] and HIROSHI ARISAWA[†]

This paper presents a new functional data model and its operators which can generate structural "object" as a semantic interface. A variety of information structures in the real world can be mapped to the database smoothly by using a set of binary relations. However, we cannot recognize structural (or hierarchical) objects in the database. The proposed operators can generate various types of complex objects from the database.

1. はじめに

近年，データベースシステムは CAD，フルテキスト処理，音声・画像処理などいわゆるマルチメディア情報を扱う分野や意思決定，知識処理支援などの高度情報処理の分野へと応用範囲を広げている．このような状況の中で，これまでのデータベースの考え方の基盤となっていた関係データモデル (relational data model) やネットワークデータモデル (network data model) では，素データが基本的に文字や数値に限定されており，しかもデータの構成が単純なものだけを取り扱うことを前提としていたことから，新しい利用分野における要求に十分に対応できないことがしばしば指摘されてきた．

このような問題を解決するための1つの手法として最近では，データベースシステムへオブジェクト指向の概念を導入したオブジェクト指向データベース (object-oriented database) が盛んに議論されている[6),7),9)]．その最大の特徴は，複雑な構造を持つデータ（しばしば複合オブジェクト (complex object) と呼ばれる）をデータベースにおいて内部構造を持ったオブジェクト (object) として直接表現することにある．さらにデータに対する操作もオブジェクトと捉え，データの振る舞いをも含めた形でモデリングを行うことができる．

しかし，データベース中のデータが内部構造を持つことは，複合オブジェクトを表現することは容易になる反面，データの中立性には大きな問題を抱えることになる．すなわち，オブジェクト指向データベースではオブジェクト自身がある階層型に構造化されているため，データベースのユーザは，データベース中と同じ形式でデータを利用することはできても，それとは異なった見方・形式でデータを利用することが難しくなってしまう．このことは，大量のデータを多数の

† 横浜国立大学工学部電子情報工学科
Division of Electrical and Computer Engineering, Faculty of Engineering, Yokohama National University

ユーザがさまざまな形式で利用することを目指しているデータベースの根本的な目的から考えれば非常に重大な問題である．実際，最近のオブジェクト指向データベースでは，個々のオブジェクトを比較的小さなものとして設計し，上位のレベルのオブジェクトが部品となる他のオブジェクトを参照するような設計も多く行われるようになっている[8]．確かにこのようにオブジェクトを小さくして，さまざまな形のオブジェクト間参照を作ることにより，多様なデータ操作に対応しやすくすることができる．しかしそれではユーザにとって明示的でない構造を作ることになり，オブジェクト指向の良さは失われてしまっている．

そこで，本稿ではオブジェクト指向とは異なった立場から次世代データベースのためのデータモデルを議論する．以下の議論は関数型データモデル (functional data model)[1)~4)] の考え方に基礎を置いている．関数型データモデルでは，内部構造を持たない『もの』を表す主体 (entity) と主体間の二項関係を表す関数 (function) だけでデータをモデリングし，それ以上の複雑な構造をデータベース中に持ち込まない．したがって，データベースの基となる構造はきわめて単純な形となる．当然，そのままでは複雑な構造を持つ複合オブジェクトを取り扱うことはできないが，データ検索の際にデータを構造化して取り出す機構をデータベースへ付加し，構造化された検索データを作り出すことができれば，データベース中から自由な形の複合オブジェクトを作り出せる可能性があり，オブジェクト指向の最大の欠点である，データベース設計者によって固定化された構造しか扱えないという問題点を解消できる．

本稿ではこのような観点から，階層構造化されたデータを記述するための記法（主体木式）を定義し，データ集合間の関数（二項）関係で構成される関数型データベースから主体木を生成するための構造化オペレータについて考察する．

2. 関数型データモデルと主体木

本章では以下の議論の前提となる関数型データモデルの基本概念について述べた後，階層構造を持つデータを記述するための主体木を定義する．

2.1 関数型データモデル

2.1.1 関数型データモデルの基本概念

関数型データモデル[1)~5)]では基本的に主体 (entity) と関数 (function) の概念だけでデータベースが構成される．主体は，実世界中において事物・事象として認識されるすべての『もの』である．主体は実世界中に存在する『もの』のデータベース世界における代理物としての点であり，オブジェクト指向データベースにおけるオブジェクトと異なり内部構造を持たない．また，ある共通の意味を表す主体の集まりを主体集合 (entity set) と呼び，主体集合につけられた名称（すなわち，個々の主体を総称するための名称）を主体型 (entity type) と呼ぶ．

関数はこの主体間の対応付けを表現する．本稿で述べる関数型データモデルの関数は与えられた個々の主体を主体の集合へ写像する．

2.1.2 関数型データモデルを用いた現実世界のモデリング

本項では，現実世界の複雑な情報がどのようにして主体と関数だけで構成される単純な世界へモデリングされるかを簡単に説明する．

図 1 は企業の従業員に関するデータの関数型データベースのスキーマをダイアグラムで表した例である．図中の長方形は『主体型 (entity type, 共通の意味を表す主体をひとまとめにしたもの）』を表し，矢印は『関数』を表している．矢印の根元側の長方形はその関数の定義域 (domain) であり，矢印の先端側は値域 (range) である．また，矢印上の文字列はその関数の関数名を表している．

このデータベースにおいて『ある従業員の氏名が“横浜太郎”である』という現実世界の情報は，**図 2** のように『従業員を表す主体 e_1 が関数 *Ename* によって文字列“横浜太郎”に写像される』という形で表現される．ここで，e_1 は現実世界における“横浜太郎”という名前の従業員の存在自体を表現している特別な主体である．このような『もの』の存在自体を表す主体は本来文字や数値では表現することができない抽象的なデータ（表示不可能なデータ (non-printable

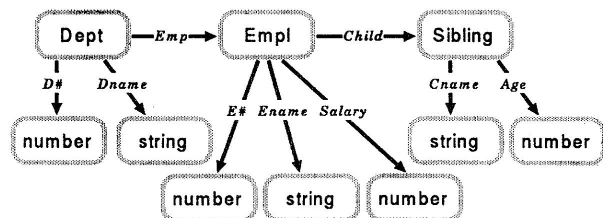


図 1　従業員データベース（スキーマ）
Fig. 1　A department-employee-child database (schema).

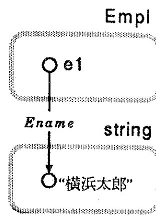


図 2　従業員"横浜太郎"の表現
Fig. 2　A representation of an employee "Taroh Yokohama."

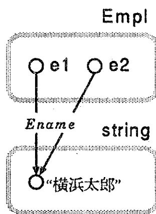


図 3　同姓同名の従業員の表現
Fig. 3　A representation of employees of the same name.

data）と呼ばれることもある）である．しかし，表現できないままでは不便なので，本稿ではこのような表示不可能な主体を e_1, s_2 といった記号を使って記述する．ある1人の従業員の存在を表す主体はデータベース中でも唯一であり，複数の主体によって1つの『もの』が表現されることはない．

また，データベース中に"横浜太郎"という同姓同名の従業員が複数存在する場合には**図 3** のような形で表現される．図では従業員のデータベース中における存在を表す主体が *Ename* によって同一の"横浜太郎"に写像されており，e_1, e_2 で表現されている2人の従業員の氏名はともに"横浜太郎"となっている．しかし，氏名を表す文字列は同じでもデータベース中における主体が e_1, e_2 と異なっていることから，この2人の従業員は同姓同名であるが全く異なる従業員であることがわかる．

図 4 は，このような関数型データモデルのモデリング手法に基づいてモデル化された従業員データベースのインスタンス（instance）を示している．また，**図 5** は図4のインスタンスを集合値を許すような関係（relation）を用いて記述した図である．本稿で取り上げる具体例はすべてこのサンプルデータベースのデータに基づいている．

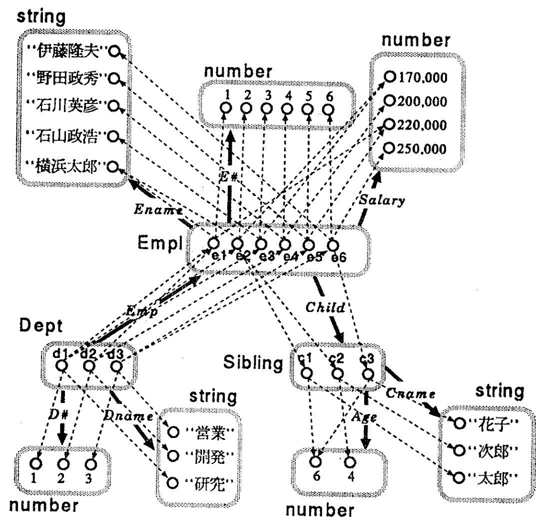


図 4　課–従業員データベース（インスタンス）
Fig. 4　A department-employee-child database (instance).

Dept

DID	D#	Dname	Emp
d_1	1	"研究"	$\{e_1, e_2, e_3\}$
d_2	2	"開発"	$\{e_1, e_4\}$
d_3	3	"営業"	$\{e_5, e_6\}$

Empl

EID	E#	Ename	Salary	Child
e_1	1	"横浜太郎"	220,000	⊥
e_2	2	"横浜太郎"	170,000	$\{c_1, c_2\}$
e_3	3	"石山政浩"	170,000	⊥
e_4	4	"石川英彦"	200,000	⊥
e_5	5	"野田政秀"	220,000	⊥
e_6	6	"伊藤隆夫"	250,000	$\{c_3\}$

Sibling

CID	Cname	Age
c_1	"太郎"	6
c_2	"次郎"	4
c_3	"花子"	6

図 5　関係表現による従業員データベース
Fig. 5　Partial contents of database in relational form (with set-valued attributes).

2.2　複合オブジェクトの表現

2.2.1　複合オブジェクトと二項関係

前述のように，関数型データモデルではすべてのデータ間の関連が二項関係で表現されているため，現実世界に存在する複雑なデータ構造もすべて二項関係に分解される．そのため，ユーザが複合オブジェクトという形でデータを利用したい場合には，ユーザは各自でデータの二項関係で構成される関数型データベースから階層構造を持ったデータで構成されるいわゆる複合オブジェクトを必要に応じて生成する作業が必要

となる．

現実問題として，ある主体が物理的に別の主体の部品になっているような場合，物理的な構造に対応してこれを表現する二項関係の集まりに対しては強い意味制約条件 (semantic constraints) が存在し，データベースシステムはその制約に基づいて検索や更新の処理を行う必要が生じるが，意味制約の表現や処理については本稿では議論しない．ここでの議論は，実世界の物理構造に対応するもの，しないものを含め関数の集合から複合オブジェクトを『いかに作り出すか』ということのみに焦点をあてる．

本稿では複合オブジェクトを『単にデータを階層的に結び付けたもの』と捉え，メッセージの内包やクラス階層などに基づく継承 (inheritance) などについても言及しない．このように，本稿で取りあげる複合オブジェクトは，オブジェクト指向データベースにおける複合オブジェクトとは若干異なるため，以下では主体木 (entity tree) と呼ぶことにする．

2.2.2 主体木の定義

本項では主体木を定義する．以下では便宜的に e, e_1, e_2, … によって主体木を表現する．

定義 1 主体木 (entity tree)

主体木は次のように再帰的に定義される．

1．数値，文字および文字列をアトム主体木 (atomic entity tree) と呼ぶ．
2．BOTTOM (⊥, undefined entity tree) は特殊な主体木である．
3．$e_1, e_2, \cdots, e_n$ がアトム主体木もしくはタプル主体木のとき，$\{e_1, e_2, \cdots, e_n\}$ をセット主体木 (set entity tree) と呼ぶ．
4．$e_1, e_2, \cdots, e_n$ が主体木で $f_1, f_2, \cdots, f_n$ $(n \geq 0)$ が関数名のとき，$e=[f_1: e_1, f_2: e_2, \cdots, f_n: e_n]$ をタプル主体木 (tuple entity tree) と呼ぶ．
5．主体木は，アトム主体木，セット主体木，タプル主体木，BOTTOM のいずれかである．

[Example]

データベースから**図6**のような階層型のデータが検索されたとする．このとき，このデータは**図7**のような主体木として表現される．

このように主体木の表現形式は，従来から階層構造のデータを表現するために伝統的に使用されてきた記述法と大きな違いはない．

2.2.3 主体木とデータベース

一般に関数型データベースではデータ間は多くの関数によって結び付けられた網目状（ネットワーク）の構造となる．この網目状のデータを主体木として木構造で表現するときに元のデータベースの構造と意味的な違いが生じることに注意が必要である．

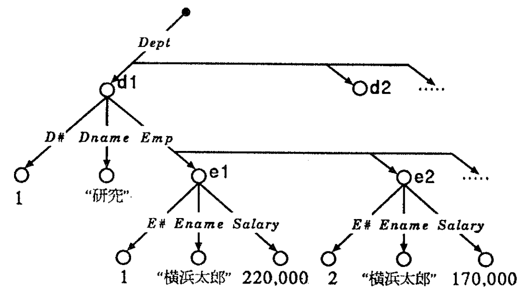


図 6 課-従業員データ
Fig. 6 Department-employee data.

```
[ dept:{ [ D#:1,
           Dname:"研究",
           Emp:{ [ E#:1, Ename:"横浜太郎",
                         Salary:220,000 ],
                 [ E#:2, Ename:"横浜太郎",
                         Salary:170,000 ],
                 … } ],
         [ D#:2,
           Dname:"開発",
           Emp:{ [ E#:1, Ename:"横浜太郎",
                         Salary:220,000 ],
                 … } ],
         … } ]
```

図 7 図6の課-従業員データを表す主体木
Fig. 7 The entity tree corresponding to Fig. 6.

具体例を考えてみよう．**図8**では，データベース中のある従業員 (e_1) が複数の課 (d_1, d_2) に所属している．データベース中では d_1, d_2 に所属している従業員は同一の従業員であるので当然，1つの主体で表現されている．

いま，このデータベースから『課データの下に従業員データが配置されている』ような構造を持つ主体木を検索の結果として生成することを考える．このとき，検索された主体木は**図9**となり，従業員データを表しているノード (node) （図9中のA，B地点）に e_1 が複数存在していることに注意が必要である．図8では e_1 という存在を持つ従業員は現実世界においてただ1人であるので，データベース中においても1つの主体によって表現されている．これに対し，主体木中では図9の e_1 のように本来1つの存在を表す主体が主体木中に複数存在することもありうる．これは，図9の e_1 はそれぞれ d_1 に所属する e_1 と d_2 に所属する e_1 に対応しているためで，主体木中のそれぞ

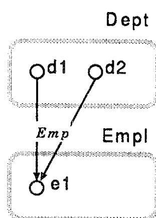


図 8 従業員が複数の課に所属する場合
Fig. 8 A case where an employee belongs to two departments.

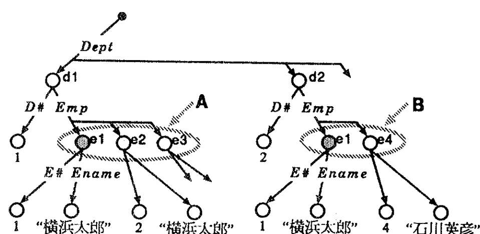


図 9 図8の課-従業員データを表す主体木
Fig. 9 The entity tree corresponding to Fig. 8.

れの e_1 が表している意味は異なっているのである．しかし，このことは e_1 で表される従業員が実際には1人しか存在しないという意味が失われてしまうことを意味するものではない．データベースに同じ主体を（例えば主体 ID のようなもので）同一と識別する機構があれば，同じ e_1 という主体であることから，同一人物であることを主体木中においても知ることはできる．

3. 主体木と検索処理

3.1 主体木とランク記述子

本節では，定義1で定義された主体木に対して，次のような制約条件を仮定する．

条件 1 主体木のインスタンス

ある1つの主体木中において同一ランクにある部分主体木はすべて同種の構造を持つ．

この条件は，『データベースから生成される主体木はすべて同じ形式の均一な構造である』ことを意味している．このような仮定を設定する理由は，検索データを利用するプログラム等において主体木を取り扱いやすくするためである．例えば，カード型インタフェースを考えたとき，データの形が共通であれば，カードの特定の位置には（例えば，画像なら画像といったように）必ず同じようなデータが配置できることが保証される．ユーザインタフェース・プログラムなどでデータベースから検索されたデータを利用する場合に，検索データの形式が1つ1つ異なっていると，プログラム側で非常に繁雑な処理が要求されることになる．

ところで，条件1中で使用している語は，以下のように定義される．

定義 2 部分主体木 (partial entity tree)

1つの主体木 E における根 (root, 図6の黒丸) 以外のノード（主体木）を根とするような主体木を『主体木 E の部分主体木』と呼ぶ．

定義 3 レベル (level)

ある部分主体木の主体木中における深さを『レベル (level)』と呼ぶ．レベル数は，根 (root) の部分から目的の部分主体木までに経由する属性の数を表す．例えば，図6において，部分主体木 d_1, d_2 は第1レベルであり，従業員名を表す文字列である部分主体木 "横浜太郎" は第3レベルである．

定義 4 ポジション (position)

あるタプル主体木においてその要素となっている部分主体木の位置を『ポジション (position)』と呼ぶ．すなわち，ポジションはタプル中における主体木の位置を表すもので，例えば，図6において，部分主体木 d_1, d_2 は第1ポジション（レベルは1）であり，従業員名を表す部分主体木 "横浜太郎" は第2ポジション（レベルは3）である．

定義 5 同一ランクの位置

いま，部分主体木 e_1, e_2 が，それぞれ（第 l_1 レベル，第 p_1 ポジション），（第 l_2 レベル，第 p_2 ポジション）に位置しているとする．ここで，$l_1=l_2$, $p_1=p_2$ が成立するとき，2つの主体木 e_1, e_2 は『同一ランクにある』という．例えば，図6において，部分主体木 d_1, d_2 はともに（第1レベル，第1ポジション）の主体木であり，同一ランクにある．しかし，従業員名を表す部分主体木 "横浜太郎" (第3レベル，第2ポジション) と従業員番号を表す部分主体木 "1" (第3レベル，第1ポジション) は（レベルは等しいが）ポジションが異なるために同一のランクに位置していない．

定義 6 同種の構造

2つの部分主体木 E_1, E_2 が与えられ，次のいずれかの条件を満足するとき，この2つの主体木は『同種の構造を持つ』という．

1．部分主体木 E_1, E_2 が，ともにアトム主体木である．

2．部分主体木 E_1, E_2 が，ともにセット主体木である．
3．部分主体木 E_1, E_2 が，ともにタプル主体木 $E_1=[A_1: a_1, \cdots, A_n: a_n]$, $E_2=[B_1: b_1, \cdots, B_m: b_m]$ であり，かつ $n=m$, $A_i=B_i$ $(i=1, \cdots, n)$ が成立する．
4．部分主体木 E_1, E_2 の一方，もしくは両方が BOTTOM である．

一般に，オブジェクト指向データベースなどではデータベース中の個々のデータ（オブジェクト）はそれぞれ異なる構造を持つことが許されている．主体木に対して条件1のような制約を付加すると，一部のデータにだけ特別な属性値が存在するなどデータの構造が必ずしもすべて同一でないようなケースの場合に問題が生じてしまうと思われるかもしれない．本稿の関数型データベースでは，主体に対して無意味な関数を適用した結果は常に未定義主体(⊥)となる．この性質を利用して，構造が異なるデータやある属性が他のデータでは定義されていない場合には，主体木中のすべてのデータに対して強制的に属性値が⊥であるような属性を付加して，結果的にすべてのデータが共通の構造を持つようにすることができる．

主体木のインスタンスが共通の構造を持つために，例えばユーザは『主体木中の関数名 “Dept” のインスタンス（実際にはタプル主体木となる）中の関数名 “D#” の各主体木に対して…という操作を行う』と記述するだけで主体木に対する操作を記述できることになる．本稿では主体木に対して操作を行う際に操作対象のデータを指定する方法として，主体木の根 (root) の部分からの関数名を列挙することを採用する．例えば，前述の例は『/Dept/D# に対して…』と記述される（**図 10** 参照）．このように操作対象の主体木を指定するために関数名を列挙した文字列を『ランク記述子 (rank descriptor)』と呼ぶ．

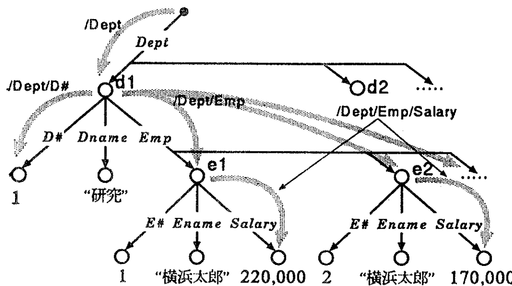


図 10 主体木とランク記述子
Fig. 10 An entity tree and rank descriptors.

ここで，ユーザは1つのランク記述子によって主体木中の複数の主体を一度に指示することができる点に注意が必要である．例えば，図 10 の主体木に対して，/Dept/D# というランク記述子によって，すべての課 (Dept) データの属性 D# の値を指定することができる．また，ランク記述子によって指定される主体群はすべて同一の関数の値域となることが保証されている．

3.2 構造化オペレータ

本章では，関数型データベースから検索される任意の階層構造をもつデータを表現する主体木に対するオペレータを定義する．

3.2.1 オペレータとランク記述子

前述したように本稿で議論の対象とする主体木のインスタンスの構造はすべて同じである．そのため，ユーザは『主体木中の全従業員に対して…という操作を行う』という形で操作を記述するだけで主体木に対する操作を記述することができる．ここで，有用となるのがランク記述子である．なぜなら，1つのランク記述子によって指定される主体木中のデータは主体木中における同一の位置に存在することが保証されており，このことは1つのランク記述子によって指定される主体はすべてある特定の関数を適用した結果として得られる主体だからである．そのため以下で定義するオペレータは，主体木中のどの主体に対する操作であるかを明示するだけでデータに対して曖昧性なく操作を行うことができる．

書式

ランク記述子 R によって指定された主体木中のすべての主体に対して，操作オペレータ *operator* を適用することを次のように表記する．

Foreach (R)
  operator ;
endfor

3.2.2 主体木に対する操作

ユーザの意図する主体木を生成／操作するために，本稿で議論する関数型データモデルでは，*generate*, *apply*, *select*, *project* という4つのオペレータを提供する．

generate は指定された主体集合名に属するすべての主体の集合を生成する．*apply* は，ランク記述子で指定された主体に対して与えられた関数をアプライし，その結果として得られた主体を主体木内へ埋め込む操作である．ユーザはこの操作により，データベー

スから検索される主体木をより複雑な構造を持つ主体木へ成長させることができる．前述の2つのオペレータにより作られた主体木中から必要な一部のデータだけを取り出すオペレータが *select* である．*project* は指定されたタプルの中から特定の属性を削除する操作である．主体木データに対する各オペレータの操作を記述したものを図 **11** に示す．

[Example]

課データ (Dept) を基点とし，その属性として課番号 (D#) と従業員データ (Emp) が並べられ，さらに Emp データの属性として従業員番号 (E#)，氏名 (Ename)，給与 (Salary) があるようなデータを検索することを考える。ただし，Emp データのうち，給与が 20 万円未満の者は検索データから除外する．

```
generate (Dept)
Foreach (/Dept)
  apply (D#)
  apply (Emp)
  Foreach (/Dept/Emp)
    apply (E#)
    apply (Ename)
    apply (Salary)
    select (/Dept/Emp/Salary
             ≧200, 000)
  endfor
endfor
```

本稿で議論する関数型データベースでは，以上の4つのオペレータを組み合わせて作られたもののみを主体木と呼ぶ．そして，これらのオペレータを使って操作された主体木は必ず条件1を満足することが保証されている．

ところで，本稿で提案している主体木は従来のデータベースにおけるビュー (view) の概念に相当する．よって，主体木に対する操作/変更/削除はその主体木に対してのみ影響を与えるもので，データベースに格納されているデータに対しては何ら影響を与えない．

3.2.3 主体木の生成手順

本項では，データベースからどのような手順で主体木が生成されていくかについて 3.2.2 項の例を用いて述べる．まず，データベースから主体木を生成する際に最初に作り出されるのは，その主体木の中心となる主体の集合を表すセット主体である (図 **12**(a)参照)．このようにし

$generate(E)$: NULL $\rightarrow$ $[E:\{e_1, e_2, \cdots, e_n\}]$,
　ただし $e_i \in E$ (Eは主体型)

$$apply(F):\quad e_i \rightarrow \begin{cases} [F:\{f_1, f_2, \ldots, f_n\}], & \text{関数}F\text{が多値関数}\ F(e_i)=\{f_1, f_2, \cdots, f_n\}\ \text{のとき} \\ [F:f_i], & \text{関数}F\text{が単値関数}\ F(e_i)=f_i\ \text{のとき} \end{cases}$$

$$select(C):\quad e_i \rightarrow \begin{cases} e_i,\ \text{ただし，条件式}\ C(e_i)=True\ \text{のとき} \\ \perp,\ \text{ただし，条件式}\ C(e_i)=False\ \text{のとき} \end{cases}$$

$project(F_i)$: $[F_1, F_2, \ldots, F_n] \rightarrow [F_1, \ldots, F_{i-1}, F_{i+1}, \cdots, F_n]$
　ここで，$F_{i(i=1\sim n)}$ は関数名とその値の組み $f_i:\{e_1, \ldots, e_n\}$ を表す

図 11 構造化オペレータ
Fig. 11 The operators.

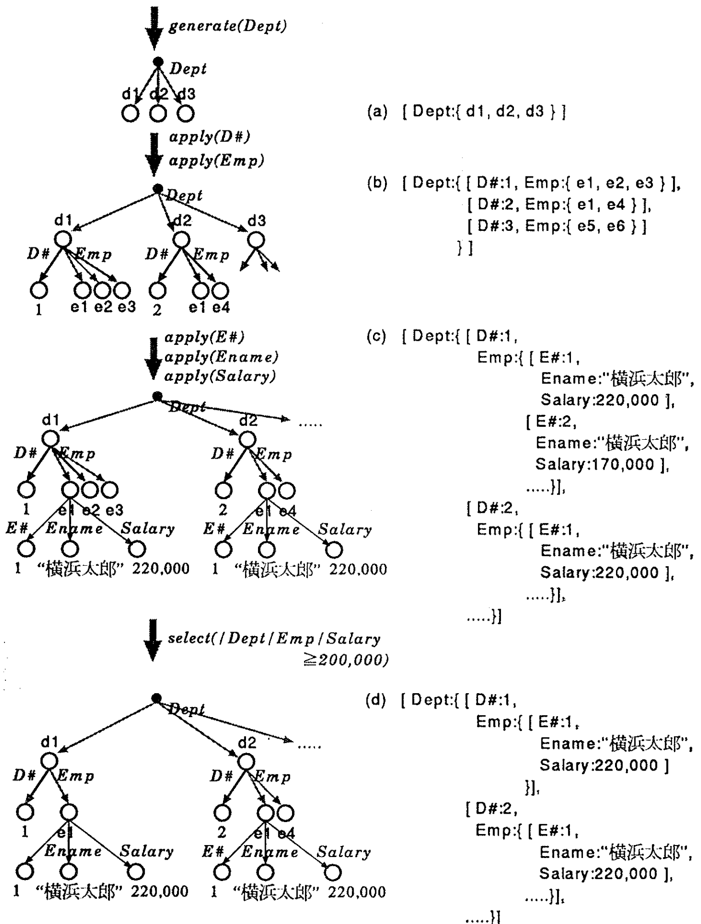


図 12 主体木生成の手順
Fig. 12 The process of generating an entity tree.

て生成された単純な１つの主体の集合を表す主体木へ個々の主体をデータベース中の関数にアプライすることよって得られる主体を埋め込むことにより，複雑な構造を持つデータ（すなわち複合オブジェクト）を表現する主体木を作り出すことができる（図 12(b), (c) 参照）．このとき，データの存在だけを表している表示不可能な抽象的な主体は，タプルとして表現され，e_1, e_2 といった記号は主体木中へ現れることはない．しかし，このことは主体木中から，e_1, e_2 といったデータの存在を表す情報が失われてしまうことを意味しているのではなく，表示不可能な抽象的な主体をタプルという形で表現しているにすぎない．

4. おわりに

本稿では，データの二項関係だけによって構成される関数型データベースに対して構造化されたデータを持つ複合オブジェクトを生成するための構造化オペレータを定義した．さらに，この構造化オペレータによって検索される複合オブジェクトを記述するための主体木をあわせて定義した．これにより，ユーザはデータベースから高度にしかも，自由な形式に構造化された複合オブジェクトを検索することができることが示された．

ただし，今回提案した構造化オペレータだけではデータベースから複合オブジェクトを検索することはできても，複合オブジェクトを用いてデータの更新を行うことはできない．この更新問題や構造化オペレータを実行する際の最適化の問題などは今後の課題である．

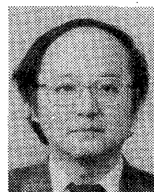
参考文献

1) Buneman, P. and Frankel, R. E.: FQL—A Functional Query Language, *Proc. of the 1979 ACM SIGMOD Conf.*, New York, pp. 52-57 (1979).
2) Buneman, P., Frankel, R. E. and Nikihil, R.: An Implementation Technique for Database Query Languages, *ACM Trans. Database Syst.*, Vol. 7, No. 2, pp. 164-186 (1982).
3) Shipman, D. W.: The Functional Data Model and the Data Language, *ACM Trans. Database Syst.*, Vol. 6, No. 1, pp. 140-173 (1981).
4) Batory, D. S., Leung, T. Y. and Wise, E. E.: Implementation Concepts for an Extensible Data Model and Data Language, *ACM Trans. Database Syst.*, Vol. 13, No. 3, pp. 231-262 (1988).
5) Arisawa, H., Nagae, H. and Mochizuki, Y.: Representation of Complex Objects on Semantic Data Model AIS and Implementation of Set Operators, *IEICE Trans.*, *E-74*, Vol. 1, pp. 191-203 (1991).
6) Bancilhon, F.: Object-Oriented Database Systems, *Proc. of ACM PODS*, pp. 152-162 (1988).
7) Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D. and Zdonik, S.: The Object-Oriented Database System Manifesto. *Proc. Deductive & Object-Oriented Databases*, pp. 40-57 (1989).
8) Deux, O. et al.: The Story of O_2, *IEEE Trans. Knowledge Data Eng.*, Vol. 2, No. 1, pp. 91-108 (1990).
9) Straube, D. D and Özsu, M. T.: Queries and Query Processing in Object-Oriented Database Systems, *ACM Trans. Office Info. Syst.*, Vol. 8, No. 4, pp. 387-430 (1990).

(平成4年10月9日受付)
(平成5年12月9日採録)

永江　尚義

昭和 41 年生．平成元年横浜国立大学工学部電子情報工学科卒業．平成３年同大大学院工学研究科博士前期課程（電子情報工学専攻）修了．現在，同大大学院博士後期課程在学中．データベースの設計理論，質問処理などの研究に従事．



有澤　　博（正会員）

昭和23年生．昭和48年東京大学理学部物理学科卒業．富士通(株)を経て，昭和 50 年横浜国立大学工学部に奉職．同学部電子情報工学科助教授．工学博士．データベース理論，データベース・システム・アーキテクチャを研究テーマとしている．著書として「データベース理論」（情報処理学会）など．電子情報通信学会会員．