データベース・システム 72－ 5
(1989. 7. 20)

意味データモデルにおける ストリーム・オペレータの導入

Introduction of Stream-operators to Semantic Data Model

有澤博 Hiroshi Arisawa
永江尚義 Hisayoshi Nagae

横浜国立大学工学部電子情報工学科
Yokohama National University

あらまし　本稿では特定の構造化を行っていない意味データベースから構造化されたオブジェクトを作り出すためのオペレータを定義する。このオペレータによりデータベースから利用者の求める構造化されたデータを取り出せるだけでなく、新たにデータを導出することができる。

Abstract　This paper proposes a framework of AIS semantic data model. The model is based on entity concept and "association", which maps an entity to a stream of entities in another set. The new operators listed here can generate structural "objects" from "flat" database.

1. はじめに

データベースの応用分野は文字や数値を中心としたデータ処理分野から図形・画像・音声データなどを含むマルチメディアＤＢに広がり、またその対象分野も記号化された事物のみを取り扱う事務分野からひとまとまりの実験データ群などファクト情報を扱う分野へと拡大している。この結果、従来のデータモデルに基づくデータベースでは、もはや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に、実世界の構造化された情報をより忠実に表現するため、意味データモデルあるいはオブジェクト指向データモデルといわれる分野ではデータの持つ意味をより直接的にデータベース構造に反映させるための議論が活発に行われるようになってきた。

オブジェクト指向データモデルはオブジェクトの持つ属性値だけでなくその操作(method)までもオブジェクトとしてひとまとめにし、オブジェクトの階層構造によって実世界の情報を表現しようとしている。また、上位のオブジェクトからmethodやデータを継承できるという特長がある。

これに対し、意味データモデルでは同様に構造情報の表現を重視するがその際主体(entity)自身は構造を持たず、主体集合間の対応づけで構造を表現する。また継承については汎化(generalization)の問題として古くから議論されている。すなわち、オブジェクト指向データモデルと意味データモデルは同じ問題に対してそれぞれの立場からの解答を与え

てきたと言える。

ここで問題となるのはオブジェクト指向データモデルではオブジェクト自身が構造化されているためデータベース中と同じ形に構造化されたデータを得ることはできるが、それ以外の見方によるオブジェクトを認識することができないことである。

一方、意味データモデルではデータベース中に構造化情報を持たないがなんらかの方法によってデータを構造化して取り出すことができればその構造化はオブジェクト指向よりはるかに自由で拡張性に富むものとなる。

本稿ではこのような観点から意味データベースに対してストリームオペレータを導入することにより、構造化されたデータを持つストリームを得ることができることを示す。

2. 意味データモデル

データモデルの歴史は構造の積極的な導入に対して肯定と否定を繰り返してきた。

まず、初期の階層モデルにおいてはデータベースの論理レコードは実世界における（内部構造を持った）「もの」そのものに対応し、階層的な構造によって情報構造を表現していた。次いで、ネットワークモデルにおいては「もの」と「もの」との間にm：nの対応（実際には2つの1：n対応）を定義することにより構造に関しては複数の見方を提供できるようになった。（例えばAというレコードタイプの内部構造としてBのレコード群があるという見方とその逆の見方である。）

ところが、関係モデルに至ってこのことは見直しを迫られた。関係モデルではデータベースは正規化されたテーブルの集まりであり、それらは相互に対等であり実世界のどのような構造を反映しているのかは少なくともデータベース自身は関知しない。従って、データベース利用者は関係の合成や操作によって目的の情報を取り出す際にデータベース作成者の「意図」を十分に察知し「正しい」構造に沿った操作を要求される。しかし、このことは「結合の罠」問題でよく知られているように必ずしも容易ではない。さらに関係モデルでは実世界の「もの」は全て記号値に置き換えて表現する。これによって記号集合に対する数学的な操作を定義しやすくなる一方、記号同士の同一性（同じ「もの」を指しているか）についての判断が困難になる。

これに対して近年「オブジェクト指向」のデータモデルが盛んに議論されるようになった。オブジェクト指向の最大の特徴は実世界の「もの」に対応して、データベース内にも内部構造を持った「もの(object)」を対応させることにある。

このことはもの自身に対する検索や内部構造を用いた高度なデータ操作（例えば図形を表示する等）を定義し易いという特長がある。しかし、データベース中に内部構造を持ち込むということは関係モデルやネットワークモデルで盛んに議論されたデータ構造の中立性については大きな問題がある。

例えば、学生,科目,成績という3つのオブジェクトタイプを考えると、学生の内部構造として科目と成績の組を持つ構造、科目の内部構造として学生と成績の組を持つ構造、各個成績から学生と科目を参照する構造の3つしかなく、どの場合でもオブジェクトと内部オブジェクトまたはオブジェクト間の相互参照を定義するために不自然さを伴う。

意味データモデルは上記の議論に1つの解答を与えている。意味データモデルの定義や考え方は提案者によって少しづつ異なるが、本質的には次のような点で一致している。

(1) 実世界の「もの」に対応させてデータベース内に記号値ではなく「もの」の代理物（これを主体(entity)と呼んでいる。）を蓄積する。

(2) 主体は内部構造を持たない。

(3) 主体集合間に方向性を持った2項の対応づけを考え、それ以上の複雑な構造を導入しない。（初期の意味データモデルではこれに反するものもある。）

2項の対応の例としてFQL[4,5],DAPLEX言語[6]の関数(function)、GENESIS[7]のrewrite ruleなどが挙げられる。

(4) 上で定義した構造に沿った検索言語を持つ。すなわち、上記の対応を組み合わせてユーザの必要とする構造を作り、そ

のインスタンスを取り出すことができる。

次節以降では意味データモデルとして著者らが提案しているＡＩＳモデルを基盤に、モデル自身が持つ中立的な構造からユーザの見方に合致した構造化された情報－これをここではオブジェクトと呼ぶ－を抽出する方法について述べる。

３．ＡＩＳモデル

ここでは、従来から提案されてきたＡＩＳモデル(1)に次の２つの変更を行ったものをＡＩＳモデルとして以下に定義する。

(1) 連想は２項に限り方向性を持たせる。

(2) 主体集合と連想に順序概念を持ち込む。

３．１　主体の定義

ＡＩＳモデルではデータベース化する実世界中において事物・事象として認識される全ての「もの」を「主体(entity)」と呼び、主体を共通の意味のもとに集めたものを「主体集合(entity set)」と呼ぶ。主体集合につけられた名前を「主体集合名」という。

また、主体集合Ａ，Ｂがあるとき Ａの１つの主体をＢの複数個の主体の列に結び付ける対応を「連想(assciation)」と呼ぶ。主体集合は連想によって定義され、主体の列は一意に順序づけられる。さらに、連想の集まりを「連想集合」、連想集合につけられた名前を「連想集合名」または単に「連想名」と呼ぶ。

３．２　連想の定義

主体集合Ａから主体集合Ｂへの連想集合は次のように定義される。

主体集合 $A = \langle a_1, a_2, \cdots, a_k \rangle$，
$B = \langle b_1, b_2, \cdots, b_l \rangle$が与えられている。
このとき、Ａ中の１つの主体に対して、Ｂ中の複数の主体を順序づけて並べたもの（これをストリームと呼び'<>'でくくって表す。）を対応させる。Ａ中の主体 a_i に対し、Ｂの主体のストリーム $\langle b_{i1}\ b_{i2} \cdots\ b_{in} \rangle$ が対応することを $a_i : \langle b_{i1}\ b_{i2} \cdots\ b_{in} \rangle$ と表記する。

例えば、Ａ中の a_1 に対応するＢのストリームが $\langle b_1, b_3, b_2 \rangle$、$a_2$ に対応するＢのストリームが $\langle b_2, b_1 \rangle$ であるとき、

$\{ a_1 : \langle b_1\ b_3\ b_2 \rangle$
$\ \ a_2 : \langle b_2\ b_1 \rangle \}$

によってＡからＢへの連想集合を表す。

ここで、１つの連想集合に対し逆方向の対応を表す連想集合として「逆連想集合」を考える。この逆連想において、Ｂの主体がＡの主体と対応づけられる順序は次の条件によって制約される。

「ＡからＢへの連想の逆連想としてＢからＡへ対応づけられるとき、Ｂの主体に対応するＡの主体の順序は必ずＡ全体の主体の順序に従っている。」

上の例では、逆連想は次のようになる。

$b_1 : \langle a_1\ a_2 \rangle$
$b_2 : \langle a_1\ a_2 \rangle$
$b_3 : \langle a_1 \rangle$

すなわち、連想集合では一方向の対応づけ(上記の例ではＡからＢ)は主体同士を実世界の意味に基づいて任意に対応づけることができ、逆は対応づけられる集合(Ａ)中の主体の順序に従って機械的につけられる。

３．３　連想集合と情報構造

３．３．１　連想集合の表記法

ＡＩＳモデルでは２つの主体集合間の対応づけを表す基本的な連想集合（２項間の連想）－主体集合Ａから主体集合Ｂへの連想－を次の形式で表現する。

@連想名：主体集合名A　対応　主体集合名B

ここで、対応とは、

< − >	1 : 1	
− >	n : 1	
< −	1 : n	
−	m : n	

の内のいずれかであり、主体集合に含まれる主体の具体値の対応を表す。

例えば、主体集合ＡからＢへの 1:n の連想集合ｆは　@ｆ：Ａ< − Ｂ　と表す。このと

き、主体集合ＢからＡへの逆連想集合ｆ'は
＠ｆ': Ｂ　－＞Ａ　と書き、ＢからＡへの
n:1 の対応となる。

３．３．２　主体集合自身の定義

ＡＩＳモデルでは主体集合自身も連想によって定義される。これは、
＠主体集合名:　＜－　主体集合名
と表記され、このときの連想名は主体集合名と一致する。

３．３．３　実定義域と包含制約

主体集合中の全ての主体が連想集合によって他の主体集合に対応づけられるとは限らない。

そこで、主体集合Ｅ中で連想ｆに関与しているＥの部分集合を主体集合Ｅにおける連想ｆの「実定義域（active domain)」と呼び、Ｅ／＠ｆと表記する。

多くのモデルでは実世界の情報をデータモデル上で表現するために様々な制約(constraint)が導入されている。（例えば、関数従属性など）ＡＩＳモデルでは主体集合間の関数従属性が連想の定義と共に指定されるが、さらに主体集合間に包含制約を考える。

包含制約とは、「ある実定義域が必ず別の実定義域に含まれる」という制約である。

例えば、ある会社においてプログラマの社員でなければＳＥになれない。ここで、プログラマに対して使用言語、ＳＥに対して経験年数という連想がそれぞれ定義されているとき、社員／＠経験年数⊆社員／＠使用言語という包含制約が存在する。

包含制約に関するその他の性質[11]は特にここでは言及しない。自明な性質として次が成り立つ。

＠ｆ: Ｅ－＞Ｆならば
Ｅ／＠ｆ⊆Ｅ，Ｆ／＠ｆ⊆Ｆ

３．３．４　ｎ項連想

実世界ではある複数の主体の組によってある主体が生み出されることがある。（例えば、時限と曜日の組から時間割のコマという主体が生成されるといったことなど）このような情報を表現するためにＡＩＳモデルでは「ｎ項連想」という概念を考える。

主体集合1,…,主体集合n-1の組によって主体集合nが生成される場合のｎ項連想は次の形式で表現される。

＠連想名1,連想名2,…,連想名n-1: 主体集合名1,主体集合名2,…,主体集合名n-1　－＞主体集合名n

ここで、連想1は主体集合1と主体集合n間の連想,…,連想n-1は主体集合n-1と主体集合n間の連想を表している。ところで、ｎ項連想は通常の２項間の連想を単に束ねたものとは異なり、次の制約条件によって制約される。

「＠ｆ,ｇ: Ａ,Ｂ－＞ＣにおいてＡのa_i,Ｂのb_jの組がＣのc_kに対応づけられているとき、Ｃ中のc_k以外の任意の主体c_lが対応づけられているＡ中の主体をa_m,Ｂ中の主体をb_nとするとき、
$a_m \neq a_i$もしくは$b_n \neq b_j$が必ず成立する。」

この条件は言い換えると、「主体集合Ａ中の１つの主体a_iとＢ中のb_jを選んだとき、その組合せが対応するＣ中の主体は高々１つである」となる。

４．ストリームオペレータ

本節では連想を表現するストリームについて述べ、そのストリーム上のオペレータを定義する。

４．１　ストリームと連想

ストリームは順序を持った主体の並びであり、このストリーム中に別のストリームを埋め込むことにより複合オブジェクト(complex object)を表現することができる。本稿ではこの複合オブジェクトのことを単にオブジェクトと呼ぶ。

オブジェクトの最も簡単なものは１つの主体集合を表すストリームである。

例えば、主体集合Ａ中の主体が$a_1, a_2, \cdots, a_n$のとき、主体集合はストリームによって$\langle a_1\ a_2\ \cdots\ a_n\rangle$で表わされる。ここで、ストリーム中の主体は主体集合中の主体の順序に従って並べられている。

複合オブジェクトはストリームの変換によって表現する。例えば、３．２の例におけるＡからＢへの連想集合は次のストリームの変換で表される。

$\langle a_1\ a_2\rangle$

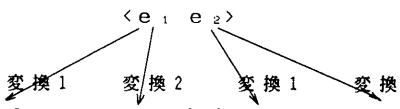
↓変換

$\langle\langle b_1\ b_3\ b_2\rangle\ \langle b_2\ b_1\rangle\rangle$

ここで、内側のストリーム$\langle b_1\ b_2\ b_3\rangle$, $\langle b_2\ b_1\rangle$は変換前の主体a_1, a_2とそれぞれ連想によって結びつけられている主体の集まり（これを「グループ」と呼ぶ。）を表し、'<>'内の主体の列は変換した主体と対応づけられていた順序を表している。このような「置き換え」による新しい構造の生成はGENESIS[7]と同様の手法である。

いま、$\langle\langle e_1\ e_2\rangle\langle e_3\ e_4\rangle\rangle$というオブジェクトを考える。ここで、$e_1$～$e_4$は従業員を表す主体とする。このオブジェクトでは$e_1, e_2$という従業員と$e_3, e_4$という従業員がそれぞれ何らかの意味（例えば所属課など）によってまとめられており、単にe_1～e_4を並べた$\langle e_1\ e_2\ e_3\ e_4\rangle$とは表現する構造が異なる。このオブジェクトにおいて、一番外側の'<>'は従業員の集まりを表し、その内側の'<>'はさらに従業員がグループ化されていることを表しており、この'<>'の深さのことを「レベル」と呼ぶ。

また、ある主体集合が複数の連想と対応づけられている場合、変換の種類が異なっているので、グループだけでは表現できない。そのためタプル（'[]'と表記する）の概念を導入する。



$\langle[\langle f_1\ f_2\rangle, \langle g_3\ g_2\rangle], [\langle f_3\ f_2\rangle, \langle g_1\ g_2\rangle]\rangle$

これは主体集合Ｅ中の主体e_1が変換１によってＦ中の主体f_1とf_2,変換２によってＧ中のg_3とg_2、Ｅ中のe_2が変換１でＦ中のf_3とf_2,変換２でＧ中のg_1とg_2にそれぞれ対応づけられていることを表わしている。

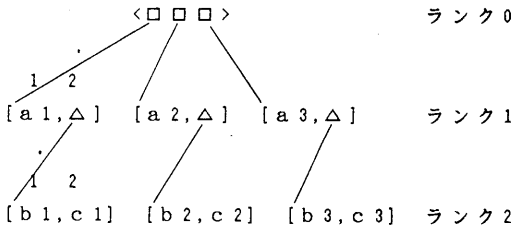
グループは同じ主体集合に属する主体の集まりである。これに対しタプルは異なった連想によって結びつけられる主体の組なので'[]'内には異なった主体集合に属する主体が並ぶことになる。

４．２　ストリームオペレータの定義

４．２．１　ランク記述子

グループとタプルの２つの構造を入れ子状に定義できるが、その中での位置を表現するためランク記述子を定義する。

いま、$\langle[\langle a_1\rangle, [\langle b_1\rangle, \langle c_1\rangle]][\langle a_2\rangle, [\langle b_2\rangle, \langle c_2\rangle]]\rangle$というオブジェクトを考える。これは次のように構造化されたものとみることができる。



ここで、タプル'[]'の深さのことを「ランク」と呼ぶ。例えば、a_1, a_2, a_3はランク１のタプルの第１要素なのでランク記述子はe(.1)と記述することにする。（'.'はランクを表し、数字は何番目の要素かを表す。）また、b_1, b_2, b_3はもとのタプルの中のタプル（入れ子のタプル）の第１要素であり、このランク記述子はランク１の第２要素のタプル（ランク２）の第１要素なのでe(.2.1)となる。

また、$\langle a_1\ a_2\ a_3\ a_4\rangle$のようなランク０のストリームに対してオペレータを適用させるときのランク記述子はe(0)と書くことにする。

'<>'と'[]'が入れ子状になっている構造では１つのランク記述子で同じレベルの複数個の主体を同時に指定できる。

４．２．２　変換オペレータ

ass(ランク記述子,連想)
tuple(ランク記述子,連想,…,連想)

変換オペレータはランク記述子によって指定された主体を連想を用いてストリームに書き換えるオペレータである。生成されたストリームには必ず'<>'を付けることにする。すなわち、ストリーム<a_1 a_2>に対し、a_1をb_1とb_2,a_2をb_3とb_4に変換するとオブジェクト<<b_1 b_2><b_3 b_4>>が得られる。ただし、主体集合を生成する連想の場合には、ランク記述子を指定しない。

ass(associateの略)は主体に対して１つの連想だけを適用し、tupleは複数の連想を１つの主体に対して適用させる。tupleオペレータによって変換されたオブジェクトには連想の組ごとに'[]'が付けられる。

４．２．３　操作オペレータ

replace(ランク記述子,値)
extend(ランク記述子,値)

いま、タプルの第１要素は部品名,第２要素は部品のサイズ（インチ）を表しているオブジェクトを考える。。
<[<'ﾈｼﾞa'>,<2>][<'ﾈｼﾞb'>,<5>][<'ﾈｼﾞc'>,<3>]>

このとき、部品のサイズの単位をインチからセンチに変更するには
replace(e(.2),e(.2)*2.5399)
とすればよい。これは、「タプルの第２要素を2.5399倍してタプルの第２要素の値と置換する」ことを表し、この結果として
<[<'ﾈｼﾞa'>,<5.1>][<'ﾈｼﾞb'>,<12.7>][<'ﾈｼﾞc'>,<7.6>]>
というオブジェクトが得られる。

このようにオペレータの'値'には算術式を書くことができる。これは操作オペレータは出力可能(printable)な主体に対してのみ適用できることを意味する。

一方、extendオペレータは求めた値をストリーム中に追加するオペレータである。

タプルの第１要素をＸ成分,第２要素をＹ成分とするオブジェクト<[<3>,<4>][<1>,<1>][<5>,<12>]>に対して、「ベクトル長を求め、それを第３要素として追加する」には
extend(e(.3),$\sqrt{(e(.1)^2+e(.2)^2)}$)
とすればよく、これにより
<[<3>,<4>,<5>][<1>,<1>,<1.41>][<5>,<12>,<13>]>
というオブジェクトが得られる。

ただし、ここで導出されたデータはデータベースに新たに蓄積されるわけではなく、仮想的に作られるものである。

GENESIS では「既にデータベース中に定義されている構造の一部を抜き出す」ことしか許されていないが、以上から明らかなようにＡＩＳモデルではオペレータにより、データベース中に存在しないデータを導出することが可能である。

４．２．４　選択オペレータ

select(条件)
project(ランク記述子)

selectオペレータはストリーム中で指定された条件を満たすものだけを取り出すオペレータである。

例えば、従業員名と年齢のオブジェクト<[<'山本'>,<20>][<'山田'>,<21>][<'山岸'>,<20>]>に対する、select(e(.2)=20)「２０才の従業員を抜き出す」の結果は<[<'山本'>,<20>][<'山岸'>,<20>]>となる。

projectオペレータはタプル中から指定された主体だけを取り出す。

先ほどのオブジェクトに対してproject(e(.1))「従業員名を取り出す」とすると、<<'山本'><'山田'><'山岸'>>というオブジェクトが得られる。ここで本来ならば、<[<'山本'>][<'山田'>][<'山岸'>]>となるはずであるがprojectオペレータはタプルの中が１つのグループだけになったときにはタプルを表す'[]'も自動的に取り除く。

４．２．５　グループオペレータ

これまに述べたオペレータはすべて「グループ」に関係なく適用されていたが、データ操作の中にはグループごとの操作を必要とする場合がある。

本節で述べるグループオペレータはオペレータとしては例外的にグループごとに適用されるものである

arrange(ランク記述子)

unique(ランク記述子)

4.2.2で述べたように変換された主体の集まりには必ずブラケット'<>'がつけられグループ化される。しかし、グループ分けせずに全体をひとまとめとして扱いたい場合もある。このようなときにarrangeによってランク記述子で指定されたグループ内の最も深いレベルのブラケット'<>'を削除することができる。

例えば、$\langle\langle a_1\ a_2\ a_1\rangle\langle a_2\ a_2\rangle\rangle$というオブジェクトに対して arrange(e(0)) とすると、$\langle a_1\ a_2\ a_1\ a_2\ a_2\rangle$が得られる。

uniqueオペレータは指定されたグループ内で最も深いレベルの主体の重複を取り除く。

例えば、$\langle\langle a_1\ a_2\ a_1\rangle\langle a_2\ a_2\rangle\rangle$に対してunique(e(0))とすると、$\langle\langle a_1\ a_2\rangle\langle a_2\rangle\rangle$というオブジェクトが得られる。

4.2.6 集約オペレータ

count(ランク記述子)

max(ランク記述子)

min(ランク記述子)

ave(ランク記述子)

sum(ランク記述子)

このオペレータはグループ単位に適用され、ランク記述子で指定された主体に対してcountは総数,maxは最大値,minは最小値,aveは平均値,sumは総和を求めるオペレータである。

例えば、オブジェクト<<e1 e2 e3><e4 e5>>に対してcount(e(0))を適用すると<[<3>,<e1 e2 e3>][<2>,<e4 e5>]>というオブジェクトを生成する。

4. 3 図形データベースへの応用

最後にＡＩＳモデルを使って簡単な図形データベースへの応用を考える。この例では図形を要素（三角形や台形など）の集まりとし、その要素は色とﾍﾞｸﾄﾙの組で表され、ﾍﾞｸﾄﾙはＸ方向成分,Ｙ方向成分から成り立っているとする。また、図形と要素はそれぞれ名前によって一意に識別される。このデータベースの各主体集合間の連想によって次のように定義される。

@図形 : <－ 図形

@図形名 : 図形 <－> 名前$

@図形要素 : 図形 － 要素

@要素 : <－ 要素

@要素名 : 要素 <－> 名前$

@要素色 : 要素 －> 色$

@要素ﾍﾞｸﾄﾙ: 要素 <－ ﾍﾞｸﾄﾙ

@ﾍﾞｸﾄﾙ : <－ ﾍﾞｸﾄﾙ

@Ｘ座標 : ﾍﾞｸﾄﾙ<－> Xmove$

@Ｙ座標 : ﾍﾞｸﾄﾙ<－> Ymove$

ここで、主体集合名の末尾の'$'はその主体集合が出力可能であることを表す。

ダイアグラムは図１のように表わされる。

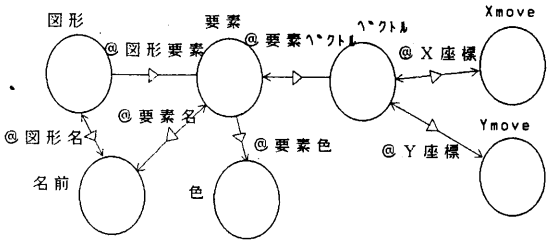


図１ 図形データベースの例

いま、このデータベースに対して次のような問合せ(query 1.～3.)を考えた場合、ストリームオペレータは次のように記述される。

Query 1. 要素色がREDである要素を含む図形名を全て求めよ。

ass(,@図形).tuple(e(0),@図形名,@図形要素).ass(e(.2),@要素色).select(e(.2)='RED').project(e(.1))

Query 2. Ｘ座標が0<x<1,Ｙ座標が0<y<2を満足するようなベクトルによって構成されている要素を含む図形名を求めよ。

ass(,@ﾍﾞｸﾄﾙ).tuple(e(0),@要素ﾍﾞｸﾄﾙ',@Ｘ座標,@Ｙ座標).select(0<e(.2)<1∧0<e(.3)<2).ass(e(.1),@図形要素').ass(e(.1),@図形名).project(e(.1)).arrange(e(0)).

arrange(e(0)).arrange(e(0)).unique(e(.1))

Query 3. 全周が50以上の要素を含む全ての図形に対してX座標を2倍に拡大したものの図形名とX座標,Y座標を求めよ。

ass(,@図形).tuple(e(0),@図形名,@図形要素).ass(e(.2),@要素ﾍﾞｸﾄﾙ).tuple(e(.2),@X座標,@Y座標).extend(e(.2.3),√(e(.2.1)²+e(.2.2)²)).sum(e(.2.3)).select(e(.2.1)≧50).replace(e(.2.2.1),e(.2.2.1)*2).project(e(.1),e(.2.2.1),e(.2.2.2))

ここで、query 1.を例に取ってオペレータを適用させていくことによってどのようなオブジェクトが生成されていくかを順を追って示す。

① ass(,@図形)
② tuple(e(0),@図形名,@図形要素)
③ ass(e(.2),@要素色)
④ select(e(.2)='RED')
⑤ project(e(.1))

① <f_1 f_2 f_3>
② <[<'図形1'><e_1 e_2>][<'図形2'><e_2 e_3>][<'図形3'><e_3 e_1>]>
③ <[<'図形1'><<'BLUE'><'RED'>>][<'図形2'><<'RED'><'YELLOW'>>][<'図形3'><<'YELLOW'><'BLUE'>>]>
④ <[<'図形1'><<'RED'>>][<'図形2'><<'RED'>>]>
⑤ <<'図形1'><'図形2'>>

5. むすび

本稿ではデータベース内に固定化した構造情報を持たない意味データベースに対してストリームオペレータを適用することにより、構造化された構造を表すストリームが得られることを示した。これにより、データベースにオブジェクトの構造を持ち込む必要性の無いことが明かになった。

今回はこのオペレータの健全性については特に議論しなかったが、オペレータをどう組み合わせても整っていないオブジェクト（例えば、<<a> b>や<<a>[<b>,<c>]>など）を生成しないなどの性質は既に明かとなっている。

またオブジェクト指向データモデルにおける操作(method)の継承のような概念は今回提案した連想だけでは表現できないが、これは導出連想(derived association)によって表現が可能となるものと思われ、これについては今後定式化を進めて行きたいと考えている。

参考文献

1) ARISAWA,H.,and MIURA,T.:"On the properties of extended inclusion dependencies.",proc.VLDB 1986

2) 有澤博 et al.:"データベースの設計・構築における意味論データモデルの利用",信学技報vol.86,No.57,(1～8)

3) 有澤博 et al.:"関数概念に基づくオブジェクト指向データモデル",電子情報通信学会,DE87-9,(9～15)

4) Buneman,P.,and Frankel,R.E.:"FQL-A functional query language.",In Proceedings of the 1979 ACM SIGMOD Conference.ACM New York,1979,52-57

5) Buneman,P., Frankel,R.E.,and Nikhil,R.:"An implementation technique for database query languages.",ACM Trans.Database Syst.7,2(June 1982),164-186

6) Shipman,D.:"The functional data model and data language DAPLEX.",ACM Trans.Database Syst.6,1(Mar.1981),140-173

7) Batory,D.S.,Leung,T.Y.,and Wise,T.E.:"Implementation Concepts for an Extensible Data Model and Data Language.",ACM Trans.Database Syst.13,3(Sept.1988),231-262