データベースシステム 104−31
（1995. 7. 20）

# ビュー機能を用いた データベーススキーマの操作

村田 美友紀　　掛下 哲郎
佐賀大学理工学部情報科学科

連続/反復/選択構造を基本とするデータモデル上でスキーマの操作を行なうための基本的枠組を提案する。この枠組は、我々が先に提案したビュー機能に基づいている。ビュー機能を用いた操作は、複数のビュー間での操作(移動、追加、コピー)を定義することで、検索機能に加えデータベースの変更も可能にしている。本稿では、ビュー機能を拡張することでスキーマ検索、変更、および複合オブジェクトの再構成(インスタンスの合併および分割)を宣言的に記述する。現在、一般的に用いられているスキーマ進化の手法はバージョンモデルを用いたものであるが、ビュー機能を用いてこのモデルを表現できることも示す。

# Retrieval and Manipulation of Database Schema using View Function

Miyuki Murata　　Tetsuro Kakeshita

Department of Information Science, Saga University
Saga 840, Japan

We propose a framework to manipulate the database schema of a fundamental data model composed by tuple, iterative and selective nodes. The framework is an extension of the view function which we have introduced. Along with the data retrieval function, the view function can update the database by defining couple of operations between two views. The extended view function enables schema retrieval and modification as well as the reorganization of complex objects. Our framework also supports the version model of schema evolution.

# 1 はじめに

データベースにおけるスキーマ変更は、(1) データベース設計、(2) 応用分野の変化、(3) 性能向上などのために不可欠である。これらの変更は、データベース運用中にも必要とされる (スキーマ進化)。また、スキーマ変更によって、既存のソフトウエアが動作しなくなるような事態は避けなければならない。本稿では、筆者らが先に提案したビュー機能 [1, 2, 3, 4] を拡張して、スキーマ変更を実現するための基本的枠組を提案する。

ビュー機能は、連続/反復/選択構造を基本とするデータモデル上で定義されている。上記の基本データモデルは、RDB, 入れ子 RDB, E-R モデル DB を表現できる。更に、OODB の複合オブジェクトを表現できるため、ビュー機能によって複数のデータモデル (連合データベースを含む) を統一的に操作できる。

ビューは射影属性、選択条件の組で定義される。選択条件は領域変数を用いた論理式である。ビューは選択条件を満たすインスタンスを検索し、射影属性の値を表示する機能を持つ。従って、宣言的なデータ操作が可能である。ビュー機能を用いた DB の変更操作は、異なるビュー間でインスタンスを操作 (移動、追加、コピー、削除) することにより実行される。ビューはウインドウ、ビュー間の操作はマウス等を用いて実現できるため、ビュー機能を用いた操作は GUI に対する親和性が高い。また、ビュー機能により簡単なデータ一貫性のチェックも行なえる。

本稿では、インスタンスに対応する領域変数とビュー間操作を拡張し、ビュー機能を用いて基本データモデルのスキーマを操作するための枠組を提案する。拡張されたビュー機能では、(1) 新しいスキーマの生成、(2) インスタンスを持たないスキーマの変更/削除、(3) スキーマ間でのインスタンスの再構成 (分割、合併) が実現できる。スキーマ進化をモデル化するために、スキーマのバージョン管理を行なう方法も提案されているが、本稿のビュー機能を用いて記述できる。

Andany 等 [5] は、コンテキストバージョニングを用いたスキーマ進化モデルを提案している。4 節で示すように、このモデルはビュー機能によって表現できる。Monk 等 [6] は、メソッド呼出しを用いて複数のクラス間でのインスタンスの変換および変更の伝搬を自動的に行なう。本稿の基本データモデルはメソッドをサポートしていないが、メソッドを追加することでこの機能をサポートできる。Tresch 等 [7] は、インスタンスの再構成を避けるために、データ独立性の概念を OODB に導入して仮想クラスとしてビューを定義する。これは、仮想クラスのインスタンスをビューに属するインスタンスによって表現することで、本稿のビュー機能によって実現できる。

本稿は次のように構成されている。2 節では、基本データモデルとビュー機能を定義する。3 節ではビュー機能を用いたスキーマの検索・変更操作を定義する。4 節では共通データモデルでバージョンモデルを表現した後、バージョンモデルを用いたスキーマ進化にビュー機能を適用する。5 節ではインスタンスの再構成のための構成操作を定義する。最後に 6 節でまとめを行なう。

# 2 ビュー機能

本節では、基本データモデルを定義する。基本データモデルによって、RDB(入れ子 RDB を含む) や E-R モデル DB を表現できる。また、OODB の複合オブジェクト[1]を表現できる。さらに、この基本データモデル上でビュー機能 (ビューおよびビュー間操作) を定義する。

## 2.1 基本データモデル

基本データモデルのスキーマ $S$はノード $N$の有限集合である。ノードの構成要素は属性とリンクである。属性は基本データ型であり、リンクはノードを参照するための型である。ノードには組ノード、集合ノード、選択ノードがあり、それぞれ以下のように定義される。

- 組ノードは、属性およびリンクの集合から構成されるノードである。
- 集合ノードはただ一つのリンクから構成され、属性を持たない。
- 選択ノードは一つ以上のリンクから構成され、属性を持たない。

ただし、集合/選択ノードは、他のノードに参照されなければならない。

[1] 基本データモデルはメソッドや継承をサポートしていないが拡張することは可能である。

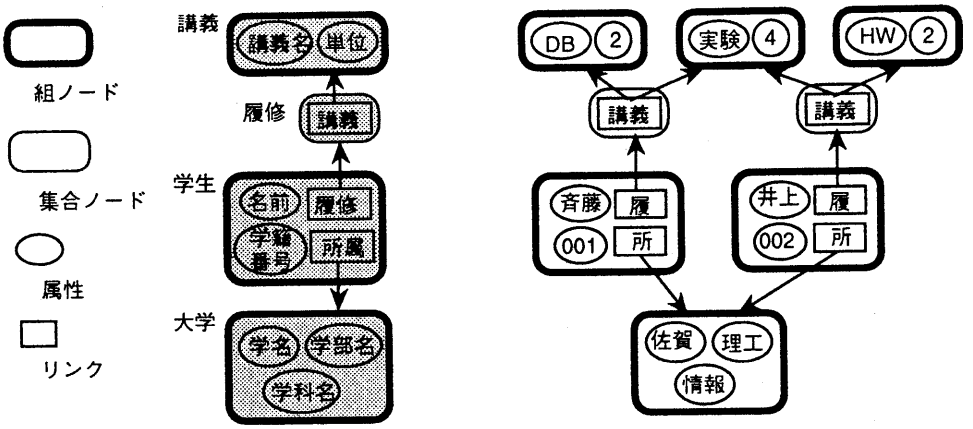


図 1: スキーマとインスタンスの例

次に、インスタンスの定義を行なう。属性の値は対応する型を持ち、リンクの値はリンクが参照するノードのインスタンスの識別子である。各ノードインスタンスは次のように定義する。

- 組ノードのインスタンスは、構成要素の属性/リンクのそれぞれの値からなる組である。
- 集合ノードのインスタンスは、$NIL$ でないリンク値の集合である。
- 選択ノードのインスタンスは、構成要素のリンクいずれか一つのリンク値からなる組である。

ただし、集合/選択ノードのインスタンスを参照するインスタンスが、少なくとも一つ存在しなければならない。

ノード $N$のインスタンス $I_N$の属性/リンク $X$の値を $I_N.X$で表す。ノードのインスタンスはそれぞれ固有の識別子を持つ。スキーマ $S$のインスタンスは、$S$に属するノードのインスタンス全体からなる集合である。基本データモデルのスキーマとインスタンスの例を図 1に示す。

## 2.2 ビュー

ビューは、基本データモデルのインスタンスを検索するために定義される。ビューの定義に先立ち領域変数を定義する。ノード $N$に対し、領域変数 $D^1_N, D^2_N, \cdots$を定義できる。$D^i_N$は $N$の任意のインスタンス $I^j_N$にマッチングできる。$D^i_N$が $I^j_N$にマッチングした時、$N$の任意の属性/ リンク $X$について

$$D^i_N.X = I^j_N.X$$

が成立する。$D^i_N$の値は $I^j_N$の識別子の値である。

ビューは射影属性と選択条件の組によって定義される。射影属性は $D^i_N.A$ で表される属性の集合であり、ビュー上に表示する属性を指定する。但しリンクは含まない。選択条件は領域変数を用いた論理式であり、インスタンスを検索するための条件を指定する。ビューの選択条件を満たし、各領域変数にマッチングするノードのインスタンスの集合を、ビューに属するインスタンスと呼ぶ。また、ビューに属するインスタンスについて、射影属性で指定された属性値から構成される組をビューに属する組と呼ぶ。ビューはそれに属する組を表示する。ビューはシステムによって識別子を割り当てられ、固有のビュー名を持つことができる。ビュー $V$の射影属性を $PA(V)$、選択条件を $SC(V)$ と表記する。また、選択条件に使用される領域変数の集合を $DV(V)$ と表記する。

## 2.3 ビュー間操作

ビュー間操作として、移動、追加、コピー、生成、削除を定義する。操作元のビュー上で組を選択することによってそのビューに属するインスタンスを(間接的に) 選択し、操作 (移動、追加等) することによって選択したインスタンスを変更できる。

**移動** 移動操作は、インスタンスの属性/リンクを

変更するために利用される。ビュー $V_s$, $V_d$を定義し、$V_s$に属するインスタンス$I$を$V_d$に移動する。この時、$D_N \in DV(V_s) \cap DV(V_d)$ を満たす領域変数$D_N$がマッチングするインスタンス$I_N(\in I)$が変更の対象となる。移動操作により、$I_N$の属性/リンク$X$の値が$SC(V_d)$を満足するように変更される。$SC(V_d)$により変更後の$I_N.X$の値が唯一に決定されない場合には、移動操作は拒否される[2]。$I_N.X$がリンクの集合である場合には、$SC(V_s)$を満たすリンク値だけが$SC(V_d)$を満足するように変更される。属性$A$を空値に設定する操作やリンク$L$を切断する操作も、選択条件$D_N.A$='undefined'または、$D_N.L$='undefined'で定義されたビューへの移動操作によって実行できる。

**追加**　追加操作は、集合インスタンスに新たなリンク値を追加するために利用される。ビュー$V_s$, $V_d$を定義し、$V_s$に属するインスタンス$I$を$V_d$に追加する。追加操作は次の二条件が満足された時にのみ実行できる。(1) $D_N \in DV(V_s) \cap DV(V_d)$を満たす集合ノード$N$の領域変数$D_N$がマッチングするインスタンス$I_N(\in I)$が存在する、(2) スキーマ中で$N$のリンク$L$が参照するノード$N'$の領域変数$D_{N'}$が、$DV(V_d)$に含まれる。$V_d$中で$D_{N'}$がマッチングするインスタンスの識別子の集合を$ID$とする。追加操作によって$I_N.L \cup ID$が新しい$I_N.L$の値になる。

**コピー**　コピー操作は任意のインスタンスをコピーして、新しいインスタンスを生成するために利用される。ビュー$V_s$, $V_d$を定義し、$V_s$に属するインスタンス$I$を$V_d$にコピーする。この時、$D_N \in DV(V_s) \cap DV(V_d)$を満たす領域変数$D_N$がマッチングするインスタンス$I_N(\in I)$がコピー操作の対象となる。コピー操作により、$I_N$と属性/リンク$X$の値が等しいインスタンス$I'_N$が生成され、$D_N$にマッチングされる。更に、$I'_N$は$SC(V_d)$を満足するように変更される。

[2]この条件を満足するために、$SC(V_d)$は以下の各条件を満足しなければならない。

- $SC(V_d) = SC_1(V_d) \wedge SC_2(V_d)$である。
- $SC_1(V_d) = \bigwedge_i (D_i.X \phi O_i)$である。ここに $D_i \in DV(V_s) \cap DV(V_d)$。$X$は属性またはリンク。$\phi$は$=$または$\ni$。$O_i$は定数、領域変数または領域変数.属性(リンク)。
- $SC_1(V_d)$は、単純値を保持する属性やリンク$X$について、その値を決定する項$(D_i.X \phi O_i)$を高々一つしか含まない。
- $SC_2(V_d)$は、$DV(V_d) - DV(V_s)$に属し、$SC_1(V_d)$中に出現する各領域変数の値を唯一に決定する選択条件である。

コピー/生成操作においても$SC(V_d)$は同一の条件を満たす。

**生成**　生成操作は、任意のインスタンスを生成するために利用される。ビュー$V$上で生成操作を実行すると、$DV(V)$の各領域変数$D_N$に対して新しいインスタンス$I_N$が生成される。$I_N$の属性/リンク$X$の値は$SC(V)$を満足するように設定される。$SC(V_d)$で指定されない属性値には$NULL$、リンク値には$NIL$ (リンク値が集合の場合は$\phi$) が設定される。

**削除**　削除操作は任意のインスタンスを削除するために利用される。ビュー$V_s$と削除操作のための特別なビュー$V_O$を定義する。$V_O$は領域変数のみで定義される。$V_s$に属するインスタンス$I$を$V_O$に移動する。この時、$D_N \in DV(V_s) \cap DV(V_O)$を満たす領域変数$D_N$がマッチングするインスタンス$I_N(\in I)$が削除の対象となる。これにより、選択したインスタンスの任意の部分インスタンスを削除できる。削除されるインスタンス$I_N$を参照するインスタンスが存在する時、削除操作を拒否する。このようなインスタンスを削除する時には、削除操作に先立ちリンクを切断しなければならない。

# 3　スキーマの操作

本節では、ビュー機能を用いたスキーマの検索およびインスタンスを持たないスキーマの編集操作を定義する。インスタンスを持つノードに対しては、参照のみを許す。このために、ノードおよび属性/リンクに対応する領域変数を定義して、各スキーマ操作にビュー間操作を対応させる。さらに、変更に伴って必要なノードの一貫性チェックをビュー機能を用いて実現する。

## 3.1　領域変数の定義の拡張

ノードおよび属性/リンクにはスキーマ情報として表1に示す特性が挙げられる。ビュー機能を用いてスキーマを操作するために、ノードにマッチングする領域変数$NS$、および属性/リンクに対応する領域変数$AS$を定義する。また、各特性は表1に示すように表記する。

## 3.2　スキーマの操作

**検索**　ノードおよび属性/リンクの検索は、ビュー定義によって行なう。ビューの選択条件で検索のた

| | 特性 | 表記法 |
|---|---|---|
| ノード | 名前 | $NS.name$ |
| | 種類 | $NS.category$ |
| | 構成要素 | $NS.type$ |
| | ラベル | $NS.label$ |
| 属性/リンク | 名前 | $AS.name$ |
| | 種類 | $AS.category$ |
| | データ型/参照ノード | $AS.type$ |

表 1: ノードおよび属性/リンクの特性

めの条件を指定し、射影属性で表示する特性を指定する。

**生成** ノードおよび属性/リンクの生成は、ビュー機能の生成またはコピー操作によって行なう。

**削除** ノードおよび属性/リンクの削除はビュー機能の削除操作によって行なう。

**特性の変更** 特性の変更は、ビュー機能の移動および追加操作によって行なう。ノードの構成要素の操作は、集合オブジェクトの操作と同様に行なうことができる。追加操作は、ノードの構成要素を追加する場合にのみ用いられる。構成要素の変更、削除は、ビュー機能の移動操作によって行なう。この時、変更の対象となる構成要素を移動元のビューで指定しなければならない。

### 3.3 スキーマ操作の一貫性チェック

スキーマ一貫性チェックを定義する前に、一貫性チェックを行なうための関数 $Count(D_N)$ と $notExist(D_N; SC)$ を定義する。$Count(D_N)$ は領域変数 $D_N$ にマッチングするノードのインスタンスの数を返す関数である。$notExist(D_N; SC)$ は、条件 $SC$ を満たす領域変数 $D_N$ にマッチングするインスタンスが存在しない時に真となる関数である。これらの関数は選択条件に含めることができる。また、$Count(D_N)$ は射影属性に含めてもよい。

各ノードが満足すべき制約を表 2に示す。一貫性を満たさない組ノード $N_T$ を検索するための特別のビュー $V_{N_T}$ を定義する。このビューの選択条件を以下に示す。

$$SC(V_{N_T}) = (N_T.category = tuple) \\ \wedge (notExist(AS^0; N_T.type \ni AS^0))$$

| ノードの種類 | 属性の数 | リンクの数 | 他ノードからの参照 |
|---|---|---|---|
| 組 | 合計 $\geq 1$ | | 必要なし |
| 集合 | $=0$ | $=1$ | 必要 |
| 選択 | $=0$ | $\geq 1$ | 必要 |

表 2: ノードの一貫性制約

ここで、$Count(N_T) \geq 1$ となるスキーマの変更操作は拒否される。ビュー $V_{N_T}$ は、スキーマの生成、変更時にシステムによって自動的に生成され、表示されないビューである。

また、集合/選択ノードには、他のノードから参照されなければならないという制約がある。この制約を満たさない集合ノード $N_I$ を検索するための特別のビュー $V_{R_I}$ を定義する。このビューの選択条件を以下に示す。

$$SC(V_{R_I}) = (N_I.category = iterative) \\ \wedge (notExist(NS, AS; \\ (NS^0.type \ni AS^0) \wedge (AS^0.type = N_I)))$$

## 4 バージョンモデルにおけるスキーマ進化

スキーマ進化には、(1) 以前の状態を保存せず、スキーマを直接変更する方法、(2) 新しいスキーマを生成し、その後にデータを再構成する方法、(3) スキーマのバージョン管理を行なう方法が考えられる。この中で最も一般的なのが、バージョン管理を行なう方法である [5]。この方法は、変更前のスキーマ上で動作していたアプリケーションの実行も保証できる。本節では、バージョン管理を用いたスキーマ進化をビュー機能によって記述する。

### 4.1 バージョンモデルの定義

基本データモデル上でバージョンモデルを用いたスキーマ進化を定義する。スキーマ上のノードはノードの種類に応じて異なるラベルを持つ。ノードを変更すると、ラベルの等しいノードが生成される。一般にスキーマ上には、複数のコンテキストバージョン (スキーマの連結した部分グラフ) が存在する。アプリケーションはいずれかのコンテキストバージョンを通して DB にアクセスするため、ス

キーマ進化後も既存のアプリケーションを実行できる。コンテキストバージョンはノードの集合(連結グラフ構造)である。また、コンテキストバージョンを構成する同一ラベルのノードは鎖状に連結していなければならない。ここでコンテキストバージョンは互いに素でなくて良い。コンテキストバージョンに対応してラベルが context である組ノードを定義する。この組ノードはコンテキストバージョンを構成するノードを参照するリンクを持つ。

## 4.2 コンテキストバージョンの操作

ビュー機能を用いたコンテキストバージョンの操作を定義する。

**ノードの追加** コンテキストバージョン $CV$へノード $N$を追加する際、スキーマ上に $N$が存在していれば $CV$に $N$を参照するリンクを追加する。存在しなければ $N$を生成し、$CV$に $N$を参照するリンクを追加する。$N$がスキーマ上に存在するか否かは、ビューを定義して $N$を検索することによって検査できる。これに加え、ビュー機能の追加操作および生成操作を用いることによって、ノードの追加が実現できる。

**ノードの除去** コンテキストバージョン $CV$からノード $N$を除去する際、$N$が他のコンテキストバージョンから参照されていれば $CV$が $N$を参照するリンクを切断する。参照されていなければ、$CV$が $N$を参照するリンクを切断した後、$N$を削除する。リンクの切断はビュー機能の移動操作、ノードの削除は削除操作によって実現できる。ここで、$N$が他のコンテキストバージョンから参照されていないことをチェックするためのビュー $V_V$を定義する。その選択条件を以下に示す。

$$SC(V_V) = (NS.type \ni AS) \wedge (AS.type = N) \wedge (NS.label = context) \wedge (Count(NS) \geq 1)$$

ここで $Count(N) \geq 1$ であれば、他のコンテキストバージョンからの参照が存在する。$V_V$はシステムによって定義され表示されないビューである。

**ノードの変更** コンテキストバージョン $CV$を構成するノード $N$の変更は、以下の手順で実行される。

1. $N$と同一のラベルを持つ新しいノード $N'$を生成し、$N'$上に以下の規則に従って属性およびリンクを追加する。(1)$N'$で新しく定義した属性を追加する。(2)$N$を含む他のノードの属性値から計算される属性 $A$を $N'$上で定義した場合には、それらのノードへのリンクを追加する。(3)$N$のリンクを $N'$でも定義した場合には、同一のリンクを追加する。(4)同一ノードへのリンクは重複して追加しない。

2. $CV$をコピーして、$N'$を含み、$N$を含まないコンテキストバージョン $CV'$を生成する。更に、$CV'$に属する各ノードが参照するノードを $CV'$に追加する。

3. $N$を根とする木構造スキーマのインスタンスを $N'$を根とする木構造スキーマのインスタンスに変換する。

1では、$N, N'$から同一の属性をアクセス可能にするために、値を一箇所で保持するための処理を行なう。2では、ノード変更に伴い新しいコンテキストバージョンを生成する。3では、ノード変更に伴うインスタンスの変換を行なう。ノード変更の例を以下に示す。

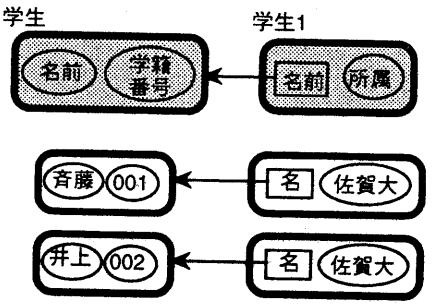


図 2: インスタンスの変換

**例 1** 図2に示すようにノード '学生' を '学生1' に変更する。この時、属性 '名前' を使用するため、'学生1' にリンク '名前' を追加して '学生' ノードを参照する。'学生' ノードのインスタンスを '学生1' ノードを根とする木構造スキーマのインスタンスに変換すると、'学生1' のインスタンスが生成され、対応する '学生' インスタンスをそれぞれ参照する。'学生1' インスタンスからの '名前' 属性のアクセスは、'名前' リンクを通じて行なうため、'学生'、'学生1' いずれのノードからも一貫した '名前' 属性をアクセスできる。なお、'学生1' ノード上に '名前' 属性を

アクセスするためのメソッドを定義すれば、リンクの存在を隠すこともできる。 □

ノードの変更はビュー機能の移動および追加操作によって実現できる。インスタンス変換を行なうための構成操作は5節で定義する。

## 5 インスタンスの再構成

4節では、コンテキストバージョンに属するノードを変更する際にインスタンスの再構成が必要であることを述べた。また、最新スキーマからのデータベースアクセス効率を向上させるために再構成を行なう場合もある。本節では構成操作を定義してインスタンスの再構成を実現する。

インスタンスの再構成に際しては、次の条件を満たすことが望ましい。(1) 単純なインスタンス変換だけでなくインスタンスの分割や複数インスタンスの合併操作を行なうこと。(2) 同一複合インスタンスを重複して生成する代りに、既存インスタンスを参照すること。

上記の条件を満足するために、ビュー上でのインスタンス生成操作を拡張してビュー間での構成操作を定義する。構成操作は、$DV(V_d)-DV(V_s)$ に属する各領域変数に対応してインスタンスを生成する。生成されたインスタンスの属性値やリンク値は $SC(V_d)$ を満足するように決定される。ただし、属性値やリンク値が全て等しいインスタンスは、重複させない[3]。

構成操作は $V_s$ 上の組を $V_d$ 上の組に変換する操作と、$V_d$ 上の組の集合を $V_d$ に属するインスタンスに変換する操作を実行することで行なわれる。このうち、$V_s$ 上の組を $V_d$ 上の組に変換するためには、$V_s$ 上の各組に対して以下の処理を行なう。$DV(V_d)-DV(V_s)$ に属する各領域変数について仮想的なインスタンスを生成し、その属性値およびリンク値を $SC(V_d)$ によって決定する。この操作によって $V_s$ 上で複数の組で表現される複合インスタンスは複数の組に分割される。

これに対して、$V_d$ 上の組の集合を $V_d$ に属するインスタンスに変換する操作は以下のアルゴリズムによって実現される。このアルゴリズムは、$V_d$ のスキーマが木構造の場合に使用できる。$V_d$ のスキーマは、$DV(V_d)-DV(V_s)$ に属する各領域変数に対応するノードとそれらの間のリンクから構成される部分スキーマである。

1. 与えられたスキーマ S において集合ノードからの経路が存在しないノードの集合をキーとする。
2. キー属性 (キーに属するノードの全属性) の値によって組の集合をグループ分けする。
3. 各グループ毎に以下の処理を行ない、生成されたインスタンス (集合を含む) へのポインタを返して終了する。
   (a) 同一グループが既に登録されていれば、対応するポインタを返す。
   (b) そうでなければ、スキーマの根ノードのインスタンスを一つ生成して、属性値をグループの値によって設定する。
   (c) 根ノードの全ての子を根とするスキーマの部分木 S' について以下の処理を実行する。
      i. 再帰呼出しを用いて、グループの組を S' に射影した組の集合をスキーマ S' のインスタンスに変換する。(根ノードが集合ノードの場合に限り、変換されたインスタンスも集合となる)
      ii. 上記で生成されたインスタンス (集合) を根ノードのインスタンスの対応するリンクによって参照する。
   (d) 根ノードへのポインタとグループを登録した後、根ノードへのポインタを返す。

構成操作によるインスタンスの分割と合併の例を以下に示す。

**例 2** 図1の { 学生, 履修, 講義 } からなる部分スキーマに属するインスタンスを図3のフラットなスキーマに属するインスタンスに変換する操作は、インスタンスの分割である。このための構成操作は、以下のビュー $V_0$ から $V_1$ に対して行なわれる。

$DV(V_0) = \{D_S$: 学生, $D_J$: 履修, $D_L$: 講義 $\}$
$SC(V_0) = (D_S$.履修 $= D_J) \wedge (D_J$.講義 $\ni D_L)$

$DV(V_1) = \{D_U$ : 受講生, $D_S$: 学生, $D_J$: 履修,

[3] 属性値が全て等しくても、異なったインスタンスを参照する必要がある場合には、同一インスタンスが重複して存在する。

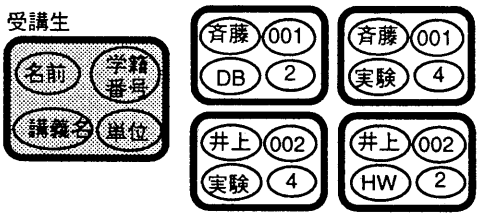


図 3: インスタンスの再構成

$D_L$: 講義 }

$SC(V_1) = (D_U.\text{講義名} = D_L.\text{講義名})$
$\wedge(D_U.\text{単位} = D_L.\text{単位})$
$\wedge(D_U.\text{学籍番号} = D_S.\text{学籍番号})$
$\wedge(D_U.\text{名前} = D_S.\text{名前})$

逆に、図3のインスタンスを合併して図1のインスタンスを構成する操作は、以下のビュー $V_2$から $V_3$に対して行なわれる。

$DV(V_2) = \{D_U : \text{受講生}\}$
$SC(V_2) = true(D_U)$

$DV(V_3) = \{D_U$: 受講生, $D_S$: 学生, $D_J$: 履修,
$D_L$ : 講義 }
$SC(V_3) = (D_U.\text{講義名} = D_L.\text{講義名})$
$\wedge(D_U.\text{単位} = D_L.\text{単位})$
$\wedge(D_U.\text{学籍番号} = D_S.\text{学籍番号})$
$\wedge(D_U.\text{名前} = D_S.\text{名前})$
$\wedge(D_S.\text{履修} = D_J) \wedge (D_J.\text{講義} \ni D_L)$ □

以上の説明から明らかなように、構成操作はインスタンスの合併と分割を一つの操作に統合している。また、重複するインスタンスの生成を行なわない。なお、上記アルゴリズムは、$V_d$のスキーマが連結な木構造または非巡回有向グラフの場合にのみ適用できる。スキーマが連結でない場合には、連結成分毎に独立した処理を行なえば良い。なお、巡回構造のスキーマは木、リスト、グラフ構造などの複合インスタンスを表現できるため重要であるが、本稿では今後の課題とする。

## 6 おわりに

本稿ではビューおよびビュー間操作の概念を拡張することによって、基本データモデルのインスタンス操作だけでなくスキーマ操作も行なえることを示した。また、バージョンモデルを基本データモデルで表現することによって、バージョン管理を用いたスキーマ進化をビュー機能で表現した。スキーマの変更に伴ってインスタンスの再構成が必要になるが、生成操作を拡張して構成操作を定義することで対応できる。

OODB のための宣言的な操作言語として、ODMG によって OQL が提案されている [8]。OQL は複合オブジェクトへのアクセスを許す SQL タイプの言語である。これに対してビュー機能は QBE タイプの言語と考えることができる。OQL は木構造でない複合オブジェクトの生成ができない点でビュー機能に劣る。また、OQL は集合指向の言語であるが、ビュー機能は集合指向の操作だけでなくインスタンス毎の操作もサポートしている。さらに、OQL が CUI ベースの言語であるのに対し、ビュー機能は GUI ベースの操作が可能である。

今後の課題としては、ユーザーインタフェースの設計、インスタンス検索の高速化などが挙げられる。また現在、ビュー機能を用いた種々の一貫性チェックについての研究も進めている。

**謝辞** 原稿の段階で御意見を頂いた九州大学情報工学科の牧之内顕文教授に感謝します。

## 参考文献

[1] 掛下, "ビュー機能を用いた構造化オブジェクトの段階的検索と編集操作", Proc. ADBS'93, pp.93-102, 1993.

[2] 掛下, 村田, "ビュー機能を用いた RDB と OODB の操作", 情報処理学会 DB 研究会 99-3, 1994.

[3] 村田, 掛下, "ビュー機能を用いた入れ子関係データベースの操作", 電気関係学会九州支部連合大会 1227, 1994.

[4] 村田, 掛下, "ビュー機能を用いた E-R モデルデータベースの操作", 情報処理学会 DB 研究会 101-5, 1995.

[5] J. Andany, et al., "Management of schema evolution in databases", Proc. VLDB, pp. 161-171, 1991.

[6] S. Monk and I. Sommerville, "Schema evolution in OODBs using class versioning", *SIGMOD Record*, Vol. 22, No. 3, pp. 16-22, 1993.

[7] M. Tresch and M. H. Scholl, "Schema transformation without database reorganization", *SIGMOD Record*, Vol. 22, No. 1, pp. 21-27, 1993.

[8] F. Bancilhon and G. Ferran "The ODMG standard for object databases". Proc. DASFAA, pp. 273-283, 1995.