データベースシステム 94−18
（1993. 7. 22）

オブジェクト指向データベースとリレーショナルデータベースの相互利用について

石川博　　久保田和己
富士通研究所

手塚正義
金沢工業大学

ここでの課題はデータモデル，システム，及びセマンティクスの異種性の解決である。そのためにモデルプリミティブ，ビューポート及びビューからなる基本的枠組みを提案する。ユーザはそのデータモデルとスキーマをローカルなサイトでモデルプリミティブを用いて記述する。このようなプリミティブを用いた記述がビューポートを構成する。ビューポートは各サイトにおいてデータモデルとデータベースシステムにおける異種性を解決する。リレーショナルビューポートでは他サイトで定義されたスキーマ（リレーショナルでもオブジェクト指向でも）はリレーショナルスキーマとしてみなされる。同様にオブジェクト指向ビューポートではどんなスキーマもオブジェクト指向スキーマとして見られる。ローカルなサイト内ではビューを用いて意味的異種性を解決する。

On a Framework for Interoperability between Relational and Object-Oriented Databases

Hiroshi Ishikawa　　Kazumi Kubota
Fujitsu Laboratories Ltd.

Masayoshi Tezuka
Kanazawa Institute of Technology

Our objective is to resolve three types of heterogeneity -- data model, system, and semantic -- in the heterogeneous database environment. The basic framework for this objective consists of model primitives, viewports, and views. Users describe their relational or object-oriented data models and schemas locally using model primitives. Description using such primitives constitutes viewports. Viewports have a role to resolve heterogeneity in data models and database systems at local sites. At relational viewports, both relational and object-oriented schemas defined at other sites are viewed as relational schemas. Similarly at object-oriented viewports, any schema defined at other sites is viewed as object-oriented schemas. Locally, relational and object-oriented views are used to resolve semantic heterogeneity.

1. はじめに

データベースの高度応用（エンジニアリングや企業情報管理など）では，既存のデータベースを組み合わせて仕事をする必要がある。そこでは複数のデータベースがボトムアップにつくられるので，それらは同一データモデル（例えばリレーショナルモデル[DATE90]）でもシステムが異なったり，最近はオブジェクト指向データベース[ISHI91][ISHI93]が出現したので，データモデルそのものが異なったりする。またたとえ同一モデルやシステムでも，意味的に同一の情報をユーザ（データベース）ごとに異なる形式で表現する場合がある。そこでそうした異種性を解決する必要がある。分散したデータベースに対するアプローチもタイトな結合からルースな結合へ，さらに相互運用性（インターオペラビリティ）の実現という現実的な解決策へ向いつつある。こうした状況で必要とされるデータベースがマルチデータベース[LITW90]である。

ここで解決しようとする課題は3種の異種性，即ち，データモデル，システム，セマンティクスの異種性の解決であるが，特に異種モデル（リレーショナルとオブジェクト指向，従って異種システム）での意味的差異の解決を目指す。そのための基本的枠組みを提案する。しかも異種データベースにおいて自律性（サイトオートノミー）をできるだけ維持した相互運用性の樹立を行いたい。自立性の尊重とは，この場合，自サイトのデータベースがリレーショナルならリレーショナルで，オブジェクト指向ならオブジェクト指向で参照できるし，さらに意味的にもユーザ独自のビューで見ることを意味する。さらに従来技術との整合性（ビュー，SQL等）を考慮すると同時に新しいモデル（例えば拡張リレーショナルモデル[KIM92]），システム，技術（例えばオブジェクト指向ビュー[ISHI92]）が追加できる開放性も考慮する。これら機能を実現するシステムをJasmine/M（Jasmine Multidatabase System）ということにする。

Jasmine/MをDATAPLEX[CHUN90]，MERMAID[TEMP86]，CALIDA[JAKO88]のような異種データモデルに関する研究と比較する。後三者はリレーショナルモデルと階層モデルの異種性の解決に焦点を当てるのに対して，Jasmine/Mはリレーショナルモデルとオブジェクト指向モデルの異種性を解決する。異種データベースのセマンティクスに関する研究[DEMI89]は主にリレーショナルデータベースに焦点をあてるのに対して，Jasmine/Mはリレーショナルとオブジェクト指向データベースの両方の文脈で意味的な異種性を解決しようとする。マルチデータベースに対するアプローチはSuperViews[MOTR87]のようなグローバルスキーマアプローチと連邦型データベースアプローチ[HEIM85]に分類できる。両者ともリレーショナルデータベースに関する。Jasmine/Mは後者のアプローチをオブジェクト指向データベースに拡張して，前述した目標を解決する。MRDSM[LITW86]はリレーショナルデータベースにおいて統一的言語を提供することにより異種データベースを統合する。一方Jasmine/MはSQLやオブジェクト指向問い合わせ言語のようなユーザ自身の言語を利用することを許す。

最近の研究（Pegasus[AHME91]やUniSQL[KIM92]）はオブジェクト指向データベースに焦点を置く。たとえばPegasusは統一的なデータモデルと言語を提供することにより，リレーショナルデータベースとオブジェクト指向データベースを統合しようとする。Jasmine/Mは高い自律性とともに相互運用性を与える。UniSQL/Mはオブジェクト指向の特徴を取り入れた拡張リレーショナルモデルを，リレーショナルと拡張リレーショナルモデルの統合モデルとして採用する。

ここで提案するフレームワークはモデルプリミティブ，ビューポートおよびビューからなる。モデルプリミティブは異種のデータモデルとシステムを記述する。基本的なデータ構造として，フラットなリレーションとネストされたリレーション[ROTH88]を提供する。制約としては型，関係，プライマリキー，フォーリンキーおよびユーザ定義制約を提供する。テーブルにたいする操作としては導出，関数，手続きメソッドを提供する。メソッド定義と起動をゆるすよう拡張されたSQLはネストされたテーブルに対する操作も提供する。

ユーザはそのデータモデル（リレーショナルかオブジェクト指向モデル）とスキーマをローカルなサイトでJasmine/Mのモデルプリミティブを用いて記述する。このようなプリミティブを用いた記述がビューポートを構成する。ビューポートは各サイトにおいてデータモデルとデータベースシステムにおける異種性を解決する役割を持つ。リレーショナルビューポートでは他サイトで定義されたスキーマ（リレーショナルでもオブジェクト指向でも）はリレーショナルスキーマとしてみなされる。同様にオブジェクト指向ビューポートではどんなスキーマもオブジェクト指向スキーマとして見られる。ローカルなサイト内ではリレーショナルビュー[DATE90]やオブジェクト指向ビュー[ISHI92]が意味的異種性を解決するために利用される。われわれは，異種モデル，異種システムおよび異種セマンティクスの解決を一歩一歩行なうアプローチをとる。以下にこの論文の構成を示す。第2節ではデータモデルとスキーマを記述するモデルプリミティブを述べ，第3節ではデータモデルとスキーマを記述するビューポートを説明する。第4節では意味的異種性を解決するためのビューについて説明する。

2. モデルプリミティブ

2.1 基本概念

Jasmine/Mのモデルプリミティブは各種データモデルとスキーマをローカルなサイトで記述できる。リレーショナルやオブジェクト指向モデルのような異種のモデルを記述できる。もちろんこれらプリミティブはオブジェクト指向やリレーショナルデータモデルと同列のものではなく，むしろデータモデルを記述するためのメタモデルである。

モデルプリミティブはフラットなテーブルとネストされたテーブルをデータ構造として提供する。テーブルはフィールドからなる。フィールドはINTEGER，STRING，OID，INNER_TABLEなどを型として持つ。INNER_TABLE型のフィールドを持つテーブルがネストされたテーブルである。フィールドは多値，必須値，プライマリキー，フォーリンキー，関係，ユーザ定義制約を持つ。関係制約は参照するフィールドと参照されるフィールドが同一型であることを指定する。他フィールドを参照するフィールドを関係と呼び，他のフィールドを参照しないフィールドを属性と呼ぶことがある。プライマリキー，フォーリンキー，関係制約は一般にはフィールドの集合に対して定義される。ユーザ定義の関数を通した型変換を指定する

こともできる。

テーブルに関連した操作，即ちメソッドは導出，関数，手続きに分類できる。導出メソッドは既存のテーブルとフィールドからSQLを用いてフィールドを導出する。これはSQLのビューや手続き[DATE90]に対応する。関数メソッドはSQLではなく制御構造（if then elseやwhile）を用いてフィールドの値を計算する。これは埋め込みSQL[DATE90]に相当する。フィールドと同様に導出や関数メソッドはINTEGER，STRING，OID，INNER_TABLEなどを型として持ち多値のような制約を持ちうる。手続きメソッドは副作用に中心をおき，埋め込みSQLに相当する。

半順序関係（superという）はテーブルに対する制約として働く。フィールドやメソッド定義の遺伝や集合的包含関係はこのsuper関係を通して行われる。この３つの機能は本来は独立であり，ユーザは三者を組み合わせることができる。

提供する代数としては既存の技術との整合性を考慮して基本的にSQLからなる。そのSQLをメソッド起動とネスト／アンネスト操作で拡張する。ネスト／アンネストはネストされたテーブルの操作である。メソッド起動はテーブルにメソッドを導入したことで付け加える。因みにメタデータはフラットテーブルへ格納管理する。SQLの代数的拡張（拡張されたSQLをXSQLという）は以下のようにネスト／アンネストを提供する。

```
select field...
from table
unnest by field...

select field, (field, ..., ), ...
from table
nest by field
```

アンネストはネストされたテーブルからフラットなテーブルを生成する。ネストはフラットなテーブルからネストされたテーブルを生成するテーブル。

さらにXSQLではメソッドを指定できる。たとえば導出メソッドはビュー（パラメータなし）かSQLの手続き（パラメータあり）として定義する。定義はSQLに限定する。関数メソッドは埋め込みSQLとして定義する。手続きメソッドは一般的プログラムとして定義する。メソッドはXSQLにおいて以下のように起動される。

```
select F1, M(F2) from T where condition
```

ここにMはTにかんするメソッドである。メソッドは問い合わせのターゲットと条件部におけるフィールドの位置に指定できる。メソッドはパラメータ（例えばF2）を持てる。

ユーザは上記のモデルプリミティブを用いてデータモデルとスキーマを定義する。以下ではリレーショナルとオブジェクト指向データベースの両方についてプリミティブを用いたモデルとスキーマ記述について説明する。

２．２　リレーショナルデータベースのプリミティブによる記述

リレーショナルモデルに関するプリミティブ記述はSQLにもとづくシステムにつき１回だけ行なう。

```
table: flat table
field: single, type: INTEGER, STRING, ...
constraint: relationship, primary key, foreign key,
        user-defined （即ちcheck option）
method: derivation method （即ちviewとSQL procedure）
algebra: SQL
```

上記の記述はデータベース管理者が行う。

次にモデルプリミティブによるリレーショナルスキーマの記述について述べる。もちろんスキーマ記述はモデル記述に矛盾してはいけない。例えば以下のスキーマ記述がSQLスキーマから生成される。

```
create table T1
(F1 INTEGER
F2 STRING
F3 FLOAT
primary key (F1)
foreign key (F2) references (T2)
check (T1.F3 > c1) )
```

```
procedure M1
SQLCODE
F1 INTEGER
select F3*F3 into VALUE from T1 where T1.F1 = F1

T1 : flat table
T1.F1 : field, single, INTEGER, primary key
T1.F2 : field, single, STRING, foreign key,
        relationship: T1.F2 = T2.F4
T1.F3 : field, single, FLOAT, user-defined constraint (T1.F3 > c1)

T1. M1 : derivation method
SQLCODE
F1 INTEGER
select F3*F3 into VALUE from T1 where T1.F1 = F1
```

ここでスキーマ記述に指定されない項目はモデル記述に従う。

2．3　オブジェクト指向データベースのプリミティブによる記述

オブジェクト指向データベースシステム（例えばJasmine[ISHI93]）につき１回だけプリミティブ記述を行なう。

```
table: nested table
fields: single, multiple
type: INTEGER, STRING, FLOAT, ... , OID,
        INNER_TABLE, conversion (OID, NUMBER, ConvFunc)
constraint: relationship: t.OID = t'.OID,
        primary key, foreign key, user-defined constraint
method:
  derivation method: single, multiple
      TYPE: INTEGER, STRING, FLOAT, ... , OID
  function method: single, multiple
      TYPE: INTEGER, STRING, FLOAT, ... , OID
  procedure method : single
      TYPE: VOID
super: field inheritance, method inheritance, set inclusion
algebra: XSQL (すなわちSQL + nest/unnest + method invocation)
```

上記記述はデータベース管理者が行う。

次にオブジェクト指向データベーススキーマのモデルプリミティブによる記述を説明する。リレーショナルデータベースと同様にスキーマ記述はモデル記述に矛盾してはいけない。例えば以下はJasmineのスキーマ記述である。

```
C1
Super    COMPOSITE
Property INTEGER  A1
         C4       A2  multiple
Method   INTEGER  M1()
     { return self.A1 * self.A1 }

C2
Super    C1
Property C3       A3
         STRING   A4  multiple
Method   VOID     M2()
{ ...
```

```
an operation with side effects (e.g., print attribute values)
... }

C3
Super    C1
Property INTEGER   A5
         STRING    A6  multiple
Method   INTEGER   M3()
{  ...
a function manipulating attributes by using programs constructs
... }
```

XSQLによる以下の記述は上記のスキーマから生成される。

```
create table C1
(OID OID
 A1 INTEGER
 A2 OID multiple)

create table C2
(OID OID
 A1 INTEGER
 A2 OID multiple
 A3 OID
 A4 STRING multiple )

create table C3
(OID OID
 A1 INTEGER
 A2 OID multiple
 A5 INTEGER
 A6 STRING multiple )

procedure M1($OID)
SQLCODE
OID $OID
select A1*A1 from C1 where OID = $OID
```

ここにキーワードmultipleは多値フィールド（ネストされたテーブルの単純形）を示す。

```
C1, C2, C3: nested table
C1.OID: field, single, OID type, primary key
C1.A1: field, single, INTEGER
C1.A2: field, multiple, OID type, foreign key
C1.M1: derived method, single, INTEGER

C2.OID: field, single, OID type, primary key
C2.A1:  C1.A1と同様
C2.A2:  C1.A2と同様
C2.M1:  C1.M1と同様
C2.A3: field, single, OID type, foreign key
C2.A4: field, multiple, STRING
C2.M2: operation method, VOID

C3.OID: field, single, OID type, primary key
```

C3.A1: C1.A1と同様
C3.A2: C1.A2と同様
C3.M1: C1.M1と同様
C3.A5: field, single, INTEGER. USER1にアクセス権はない
C3.A6: field, multiple, STRING;super (C1, C2) 及びsuper (C1, C3)が成り立つ
C3.M3: function method, single, INTEGER

導出メソッドはパラメータがなければビューに，あればSQL手続きに変換される。スキーマ記述はオブジェクト指向モデルのプリミティブ記述に矛盾しない。

3． ビューポート

3．1 基本概念

ビューポートはサイトオートノミーを保証しながら異種モデル、異種システムの差異の解決を行うのに利用される。他サイトで生成されたデータベースを自サイトのデータベースとしてみなす。すなわちビューポートをとおして応用スキーマはユーザサイトでのスキーマにマッピング（写像）される。ビューポートはモデルプリミティブで記述されたモデル（リレーショナルやオブジェクト指向）が各利用サイトでどのように見えるかを指定する。各サイトで利用するデータモデルのモデルプリミティブによる記述の結果であるプリミティブの集合およびサイト固有の情報（ユーザ情報など）が，そのサイトでのビューポートになる。そこに到着する他サイトの個別のスキーマはビューポートというフィルターを通して見える。

ビューポートは以下のような性質を持つ。

性質１：ビューポートをとおしてのみ，サイト外のデータベーススキーマはアクセスできる（即ち異なるモデルはビューポートを通してのみ相互利用できる）。

性質２：たとえ同一モデルでもビューポートが異なれば（即ちデータモデルが異なる場合とアクセス権などのユーザ情報が異なる場合がある），異なって見える。

性質３：存在は教えてくれるが，直接サポートはしない情報の提供を行う。ユーザは工夫すればそれを利用できる。

データベーススキーマはメタデータを構成し，各サイトに送られる。そこでビューポートとしてのメタデータとデータベーススキーマとしてのメタデータは結合される。その結果できるメタデータは新しいデータベーススキーマをあらわす。データベースそのものはその新しいデータベーススキーマに基づいて解釈される。

$(MD, D) \times (MD_{VP}\ U) \rightarrow (MD', D')$

MD, MD': metadata for database schemas
D, D': databases
MD_{VP}: metadata for a viewport
U: user information

以下ではリレーショナルまたはオブジェクト指向データベーススキーマがビューポートでどう変換されるか説明する。

3．2 リレーショナルビューポート

リレーショナルおよびオブジェクト指向データベーススキーマがリレーショナルサイトで写像される。両者ともリレーショナルモデルの記述から導かれたリレーショナルビューポートにしたがってリレーショナルスキーマとして見られる。

（1）リレーショナルデータベーススキーマの場合

SQLデータベースとメタデータはリレーショナルビューポートへおくられ，そこでメタデータはビューポートのユーザ情報に基づいて修正される。修正されたデータベースはそのままである（ただしセキュリティに関する部分は異なる）。

（2）オブジェクト指向データベーススキーマの場合

オブジェクト指向データベースとそのスキーマはリレーショナルビューポートへ送られ，データベーススキーマとしてのメタデータはビューポートメタデータによってフィルタされる。例えば，２．３のスキーマは以下のように修正される。

```
create table C1
(OID NUMBER
 A1 INTEGER
```

~~A2 NUMBER multiple~~)

create table C2
(OID NUMBER
A1 INTEGER
~~A2 NUMBER multiple~~
A3 NUMBER
~~A4 STRING multiple~~)

create table C3
(OID NUMBER
A1 INTEGER
~~A2 NUMBER multiple~~
A5 INTEGER
~~A6 STRING multiple~~)

procedure M1($OID)
SQLCODE
NUMBER $OID
select A1 * A1 from C1 where OID = $OID

C1, C2, C3: flat table
C1.OID: field, single, NUMBER, primary key
C1.A1: field, single, INTEGER
C1.A2:多値なのでサポートされない
C1.M1: derived method, single, INTEGER

C2.OID: field, single, NUMBER, primary key
C2.A1: C1.A1と同様
C2.A2: C1.A2と同様
C2.M1: C1.M1と同様
C2.A3: field, single, NUMBER, foreign key
C2.A4: 多値なのでサポートされない
C2.M2: 操作メソッドなのでサポートされない

C3.OID: field, single, NUMBER, primary key
C3.A1: C1.A1と同様
C3.A2: C1.A2と同様
C3.M1:C1.M1と同様
C3.A5:USER1はアクセス権がないのでサポートされない
C3.A6:多値なのでサポートされない
C3.M3: 関数メソッドなのでサポートされない.

ここではsuperはサポートされない。

3．3　オブジェクト指向ビューポート

オブジェクト指向ビューポートではリレーショナルとオブジェクト指向データベースはともにオブジェクト指向データベースとして見える。

（1）リレーショナルデータベーススキーマの場合

リレーショナルデータベースとそのスキーマはオブジェクト指向ビューポートに送られ，そこでリレーショナルスキーマであるメタデータはビューポートを通して修正される。例えば，2．2のスキーマの修正は以下のように行なわれる。

create table T1
(OID NUMBER

```
F1 INTEGER
F2 INTEGER
F3 FLOAT
primary key (F1)
foreign key (F2)
check (T1.F3 > C1))

T1: flat table
T1.F1: field, single, INTEGER
T1.F2: field, single, STRING
T1.F3: field, single, FLOAT, user-defined constraint (T1.F3 > C1)
T1.OID: field, single, OID, primary key
```

テーブルを第一級オブジェクト（first-class object）にするためOIDが付け加えられる。super (TABLE, T1) は成り立つ。TABLEはフラットなテーブルのルートクラスである。

```
procedure M1
SQLCODE
F1 INTEGER
select F3 * F3 into VALUE from T1 where T1.F1 = F1
```

さらに以下のJasmineスキーマが上記メタデータから生成される。

```
T1
Super   TABLE
Property INTEGER F1
         STRING F2
         FLOAT F3
           Constraint (Value > c1)
Method FLOAT M1 ($F1)
        INTEGER $F1;
        { T1.F3 * T1.F3 where T1.F1 = $F1}
```

（2）オブジェクト指向スキーマの場合
スキーマはそのままである（ただしセキュリティなどのユーザ情報に関する部分は異なる）。

4． ビュー

4．1 スキーマ変換

前述のようにビューポートはデータベースシステムとデータモデルにおける異種性を解決する。しかしながら意味的異種性は同一モデル内部でさえも解決されずに残る。即ちスキーマによって意味的に同等な属性が異なる名前，表現，構造を持ったり，値がなかったりする。反対に異なる属性が同一の名前を持つこともある。そうした問題はオブジェクト指向データベースのメソッドにも起こりうる。こうした同一モデルにおける意味的差異を解消する問題に対してはビューアプローチを用いる。スキーマ変換のためにリレーショナルビュー[DATE90]やオブジェクト指向ビュー[ISHI92]を利用する。ビューの参照と更新の両方を考慮する。もっともその機能はデータモデルによって異なる。ビューはビューポート内で利用される。ビューに対するユーザの問い合わせはビューポートでサポートされるデータベーススキーマの問い合わせに変換される。

4．2 リレーショナルビュー

最も汎用的である，リレーショナルデータベースのビューを組み込む。例えば，ビューは以下のように定義される。

```
create view V1(F4,F2,SUMF3)
as select F1, F2, SUM(F3) from T3 where F1 < V1
```

リレーショナルビューポートにおいてビューV1はテーブルT3に対して作成される。F4はT3のF1に対する名前変更である。F2はT3からの選択的遺伝である。SUMF3はT3のF3からSUMを使って導出される。ビューV1に対してユーザが問い合わせすると，それはリレーショナルビューポートでサポートされるリレーショナルデータベーススキーマに対する問い合わせに変

換される。例えば以下ようにビューは変換される。

```
select F4, SUMF3 from V1
where F2 = V2
→  select F1, SUM(F3) from T3
     where F1 < V1 and F2 = V2
```

ただし当面リレーショナルビューの更新をシステムはサポートしない。なぜならリレーショナルビューを通した更新を扱う一般的なメカニズムがまだないからである。もちろんリレーショナルデータベースにおけるより進んだスキーマ変換（例えば[DEMI89]）があれば、それを組み込んで利用することは可能である。

4．3　オブジェクト指向ビュー

スキーマ変換はオブジェクト指向ビューポートにおいてビュークラス[ISHI92]をユーザが定義することで基本的に行う。ビュークラスはビューポートでサポートされるオブジェクト指向データベースのベースクラスに対する問い合わせとして定義する。例えばJasmineでは以下のようなスキーマ統合のためのビュークラス（ベースクラスの選択）を提供する。

```
VC1
Super       VIEW
BaseClass   C3
Property    A4 = A1
            A2 = A2 * 4
            A3 = A3
Method      M4 { ... }
condition   A4 < V1
```

VC1はベースクラスC3のサブセットを条件を指定して定義する。A4はC3のA1の名前を変更する。A2はC3のA4から新しく導出される。A3はC3から選択的に遺伝される。M4はビューVC1に対するメソッドである。オブジェクト指向ビュークラスに対する問い合わせはオブジェクト指向ビューポートにおいてサポートされるオブジェクト指向データベースに対する問い合わせに変換される。

```
VC1.A2 where VC1.A3 = V3
→ C3.A2* 4 where C3.A3 = V3 and C3.A4 < V1
```

ビューはメソッドのポリモルフィズム[STEF86]を利用してオブジェクト指向ビューポート内で更新できる。たとえばユーザはビュークラスに対して以下のように更新メソッドを定義する。

```
VC1
  ...
  Method
     replace (Attr, Val)
     { if Attr = A4 then C3.replace (A1, Val)
     else if Attr = A2 then C3.replace (A2, Val/4)
     else if Attr = A3 then C3.replace (A3, Val) }

     delete (Attr, Val)
     { if Attr = A4 then C3.delete (A1, Val)
     else if Attr = A2 then C3.delete (A2, Val/4)
     else if Attr = A3 then C3.delete (A3, Val) }

     insert (Attr, Val)
     { if Attr = A4 then C3.insert (A1, Val)
     else if Attr = A2 then C3.insert (A2, Val/4)
     else if Attr = A3 then C3.insert (A3, Val) }
```

ここであらかじめベースクラスには更新メソッドがシステムによって定義されているものとする。したがって更新メソッド

は，ベースクラスとビュークラスでインタフェースは同一で実現は異なるという点でポリモルフィズムを利用する。ビューはビュークラスのメソッド起動で更新される。例えば以下のビューに対する更新はベースクラスの更新に変換される。

VC1.replace(A2, V2) where VC1.A3 > V3
→ C3.replace (A2, V2/4) where C3.A3 >V3 and C3.A4 < V1

ベースクラスのサブセットに加えて，集合演算を用いてベースクラスのスーパーセットを定義することもできるが、ここでは述べない。ビュークラスはシステム定義のVIEWクラスかユーザ定義のビュークラスのサブクラスとして定義される。ビュークラスのインスタンスに関する集合的包含関係をビュークラス間では保証しない。もちろんオブジェクト指向データベースにおけるより進んだビュー機能（例えば[RUND92]）があれば，それを組み込んで利用することは可能である。

ユーザにとってのインタフェースはビューである。ユーザはビューに対して問い合わせを行う。それはビューポートでサポートするスキーマに対する問い合わせに変換される。それは他サイトへ送られ、そのサイトのデータベースに対する問い合わせに変換される。ローカルなDBMSで評価され、その実行結果はビューのインスタンスとして生成される。そのインスタンスは問い合わせを発したサイトに送られ，そこで参照される。

5．おわりに

異種データモデル（リレーショナルモデルやオブジェクト指向データモデル）からなるマルチデータベースの枠組みについて述べてきた。われわれの目的はできるだけ自律性を維持しつつ異種データベース間の相互運用性を達成することである。そのための枠組みはモデルプリミティブ，ビューポート，ビューからなり，データモデル，データベースシステム，およびセマンティクスを一歩一歩解決するアプローチをとる。今後は枠組みの妥当性の検証を行う予定である。

参考文献

[AHME91] Ahmed, R., et al.: The Pegasus Heterogeneous Multidatabase System. IEEE Computer, vol. 24, no.12 (1991).

[CHUN90] Chung, C.-H.: DATAPLEX: An Access to Heterogeneous Distributed Databases. Comm. ACM, vol. 33, no. 1 (January 1990).

[DATE90] Date, C. J.: *An introduction to database systems, volume 1*, Addison-Wesley, Reading, MA, USA (1990).

[DEMI89] DeMichiel, L.: Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains, IEEE Trans. Knowledge and Data Engineering, vol. 1, no.4 (1989).

[HEIM85] Heimbigner, D., et al.: A Federated Architecture for Information Management, ACM Trans. Office Info. Systems, vol. 3, no. 3 (1985).

[ISHI91] Ishikawa, H. et al.: An Object-Oriented Database: System and Applications. Proc. IEEE Pacific Rim Conf. Communications, Computers, and Signal Processing (Victoria, BC, Canada). IEEE, Los Alamitos, CA, USA (1991).

[ISHI92] Ishikawa, H. et al.: An Object-Oriented Database System and its View Mechanism for Schema Integration, Proc. Second Far-East Workshop on Future Database Systems (Kyoto, Japan), pp. 194-200 (April 1992).

[ISHI93] Ishikawa, H. et al.: The Model, Language, and Implementation of an Object-Oriented Multimedia Knowledge Base Management System, ACM Trans. Database Syst., vol. 18, no. 1 (1993).

[JAKO88] Jakobson, G., et al.: CALIDA: A Knowledge-Based System for Integrating Multiple Heterogeneous Databases. Proc. 3rd Intl.Conf. Data and Knowledge Bases: Improving Usability and Responsiveness (Jerusalem, Israel, June1988).

[KIM92] Kim, W.: Introduction to SQL/X. Proc. of the Second Far-East Workshop on Future Database Systems (Kyoto, Japan, April 1992).

[LITW86] Litwin, W., et al.: Multidatabase Interoperability, IEEE Computer, 12, 1986.

[LITW90] Litwin, W., et al.: Interoperability of Multiple Autonomous Databases, ACM Computing Surveys, vol. 22, No. 3 (1990).

[MOTR87] Motro, A.: Superviews: Virtual Integration of Multiple Databases, IEEE Trans. Software Engineering, vol. 13, no. 7 (1987).

[ROTH88] Roth, M. A., et al.: Extended Algebra and Calculus for Nested Relational Databases, ACM Trans. Database Syst., vol. 13, no. 4 (Dec. 1988).

[RUND92] Rundensteiner, E.A.: MultiView: A Methodology for Supporting Multiple Views in Object-Oriented Databases. Proc. 18th VLDB Conf. (1992).

[STEF86] Stefik, M., et al.: Object-Oriented Programming: Themes and Variations.
AI Magazine, vol. 6, no. 4 (Winter 1986).

[TEMP86] Templeton, M., et al.: Mermaid: Experiences with Network Operation. Proc. IEEE 2nd Intl. Conf. Data Engineering (1986).