データベース・システム 77－3
（1990．5．17）

拡張可能データベースシステムにおけるデータベース言語処理系について

小島 功　田沼 均　佐藤 豊　海老原 一郎　真野 芳久

電子技術総合研究所 情報ベース研究室

要旨

本稿では、著者らが研究、開発中の拡張可能なデータベースシステムの概要と、そのうちデータベース言語処理系の構成と実現法について述べる。このシステムは、LAN環境でマルチメディア応用を支援することを目的としており、個々の利用者がデータベースシステムの機能のうち必要とする機能を組み合わせたり、組み込むなどして好みのシステム環境を構築できるようなアプローチをとっている。これを実現するため、独立した機能部品の結合による機能分散的なシステム構成と、部品のデータベースへの格納、管理（定義情報のDB化）といった特色を持つ。言語処理系はデータベースに格納するデータに対して利用者が階層構造を持つオブジェクト型を定義できる言語で、対話的な検索を目的としSQLと同様の集合データ操作を提供する。さらに、他の言語を用いて記述、コンパイルしたプログラムもデータベースに適用できるプログラムとしてデータベース操作と動的にリンクして実行させることができる。分散環境でネットワークを介してマルチメディアデータ処理を行なうため、RPCを使ったデータベース処理機能も提供している。集合演算などのデータベース処理は大容量の主記憶装置を仮定した処理の最適化を行なっている。

Database Language Subsystem in an Open Multimedia Database System

Isao KOJIMA　Hitoshi TANUMA　Yutaka SATO　Ichirou EBIHARA
Yoshihisa MANO

Information Base Section, Electrotechnical Laboratory

Abstract

In this paper we present the design and implementation of an object-oriented database language for multimedia database applicaions. This language is based on SQL and used as an interactive query language for ad-hoc retrievals. Data definitions and data manipulations are extended to support object-oriented concepts. Since SQL lacks the power of descripting database methods, we provide an external interface to other programming languages. Database system can dynamically invoke these compiled user programs. Program interface is based on the RPC calls and it provides distributed environment supports. By using timeouts, it also guarantees system safety from errors of user database programs.

1 まえがき

マルチメディアに代表されるデータベース応用分野の拡大に伴って、データベースシステムの機能に対する利用者の多様な要求を効率的に支援することが重要となってきている。このような要求下では、データベースの表現、操作、データベースの実現環境などが応用によって大きく異なるので、利用者レベルでデータベースシステムの機能を自由に定義、拡張できることが望まれている。このようなカスタマイズ機能は、オブジェクト指向データベース [1] や、拡張可能データベースシステム [2,3] 等において様々な形で実現されようとしている。

著者らは、個々の利用者が自分の応用に適したデータベース環境を生成できることを目標とした拡張可能なデータベースシステムとして、開放型複合データベースシステム [4] を開発中である。これは、データベースシステムを構成する機能をツールキット的に部品として提供し、利用者はこれらを組合せたり、新たな部品を組み入れるなどして好みのシステム環境を構築できるようにすることを目標としている。このため、次のような性質を持つ拡張性の高いデータベースシステムを実現しようとしている。

1. **開放性** データベースシステムを構成するさまざまな機能の仕様が明らかで、相互に利用可能であること。

2. **拡張性** 機能の置き換え、追加などによるシステムの拡張は、従来部分の変更をできるだけ少なく行なえるようになっていること。

データベースシステム内部の機能が開放されていることで、利用者が必要とする機能を簡単に利用することができる。また、拡張性の高い構成により利用者は変更を繰り返し行なってシステムを容易に改良することができる。

2 開放型複合データベースの構成

以上のような性質を実現するため、開放型複合データベースシステムは、以下に示すような2つの実現上の特色を有している。

(1) データベースシステムを構成する機能部品（サブシステム）を、できるだけ独立性の高い構成要素で実現し、システムをそれらの結合により構成する。

データベースシステムを独立性の高いサブシステムの集まりとして構成することで、要求されるシステムの機能に対して負荷や機能の分散が容易に実現できる。環境の変化に伴うシステムの拡張や変更も単純となり局所化できよう。また、各サブシステムを疎に結合することで、サブシステム間の切断、結合などによる置き換えが容易になる。このようにすることで、サブシステムのテストやデバッグ等を効率良く行なえるので、未完成なシステムのエラーがシステム全体に波及することを防ぐことができる。

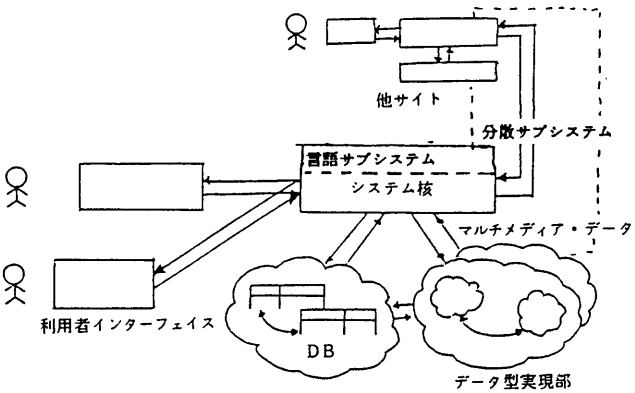
一つのデータベースシステムを例えばプロセスの結合によって実現することは、従来のシステムに比べて柔軟性の高い構成を提供できるが、一方で既存の分散データベースシステムのサイト間で実現されるほどのサブシステム同士の独立性は持たない。しかし、機能分散的なアプローチではある程度の依存性は避けられない。また、RDA のような一般的な通信方法は高速な LAN の環境ではオーバーヘッドが大きく効率的でなく、LAN 環境でマルチメディア応用を支援するにはこで提案するような構成の方が有益な場合が多いと考えられる。

各サブシステムによって実現される機能は大きく次のように分けられる。（図1）

- **核部分／問い合わせ処理部** 核は分散に伴うデータの重複などの記憶管理を行なうと共にサブシステム間のインターフェイスを管理する。例えば問い合わせ処理などで型固有の操作が必要となった場合に対応するデータ型を実現するサブシステムに処理を依頼するといった処理を行なう。

- **利用者固有のデータ型実現部** 型に固有の操作を行なうサブシステムで、核と接続のためのプロトコルが規定されている。プロトコルは分散対応にすることができ、分散ファイルシステムのようにデータベースを構築することができる。また、マルチメディア処理、記憶装置のような特殊なデバイスもこのプロトコルに従って接続することもできる。

- **利用者インターフェイス部** 問い合わせの前処理や結果の表示といった利用者固有の環境を実現するためのフロントエンドである。



（図1 開放型複合データベースシステムの構成）

以上のような環境を提供することで、利用者は固有の応用向きのデータベースシステムを実現できる。しかし、末端利用者にとってはシステムの全ての機能をいちいち定義することは労力の大きな作業であり、様々なツール、機能部品が提供される必要がある。また、多くの部品の提供される機能を把握して適切なものを選択したり、変更の場合でも既存の部品をうまく使って労力を最小限にするにはシステムによる何らかの支援機能が必要である。そのために、次のような構成を考える。

(2) それぞれのサブシステムやその接続、拡張と言ったシステムを特徴づけるような情報をデータベースに記憶、管理する（定義情報データベース）。サブシステム自身も機能部品（含プログラム、プロセス）としてその仕様などがデータベースに登録される。これらの情報はデータベース自身が管理し、サブシステムの管理はデータベース管理機能を用いて行なう。

このように、定義情報をデータベース自体が記憶し、データベース操作によって利用者環境を構築する方法には次のような利点が考えられる。

1. 定義情報自体をデータベース化することで、型の情報など環境の定義に伴う種々の概念をデータベースのスキーマとして整理できる。利用者はデータベース言語を用いてこれらを検索、更新できるので、その内容の理解、変更が容易になる。

2. 利用者の構築した環境は定義情報データベースにおける一種のビューとして実現することもでき、必要な情報を共有しながら固有の環境を実現することもできる。これにより、記憶した部品の再利用や共有による効率化が図られる。

3. 汎用のプログラム言語による記述をデータベースに記憶、実行することになるので、プログラム言語との機能の統合、相互の接続が図りやすい。

4. プログラム部品の記憶実行などの機能によって、計算によって導かれるデータや、制約条件の記述など、データベースの表現能力が高くなる。

定義情報のデータベース化によってデータベースシステムのデータ管理、検索機能を使って、多くの部品の関係の把握や既存の部品をうまく使った変更、改良などを効果的に支援する機能が提供でき、利用者の労力を軽減できると考えられる。

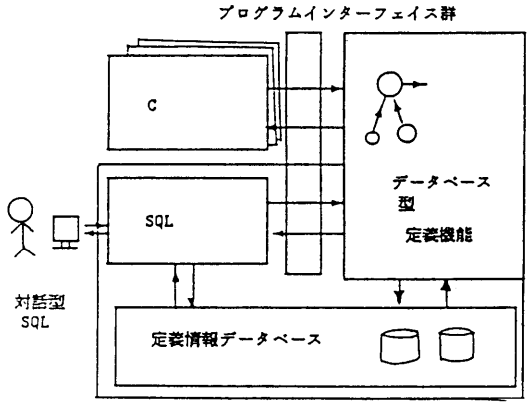
このような、サブシステムの接続形態によりデータベースシステムを実現するために、著者らは次のような問題を扱っている。

1. **データベース言語** 拡張可能なデータモデルなどを支援できる、柔軟性の高いデータベース言語

2. **機能分散環境でのデータベース処理** 分散環境で様々な機能仕様を持つシステムを接続し、データベース操作を行なうためのプロトコルとそれを実現するファイルシステム [6]。

3. **利用者インターフェイス** 利用者の応用や要求に応じて、その構成や機能が適合できるような利用者インターフェイス [14]。

定義情報データベースではプログラム等をデータベース化して格納、管理するので、データベースの格納対象としてのプログラムの管理や実行などに伴う問題も考えられており、これらの問題を扱うためのシステム環境の実現も重要となっている。



（図 2　言語処理系の構成）

3　拡張可能データベースシステムのための言語

利用者向きのデータベース環境を構築するためのデータベース言語としては、次のような機能が必要となる。

1. **定義、拡張できるデータ型の支援**：抽象データ型と同様に、利用者がデータベース化したいデータ型の表現とその操作を定義し、システム内に組込めるようにする。このことで、応用に必要なデータ表現形式やデータ操作を自由に定義、利用できる。

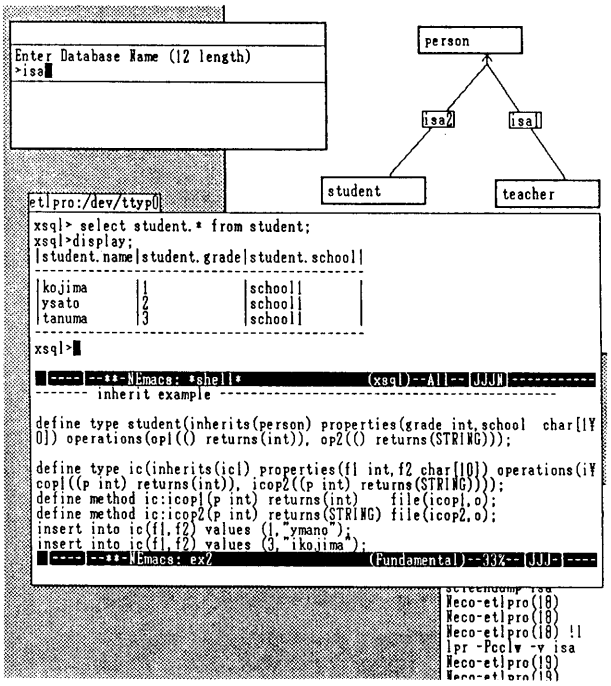
2. **データモデルの拡張性の支援**： CAD における部品展開のように、データベース化の対象によって固有の意味を持つデータベース表現要素が必要な場合がある。これを実現するには基本となるデータモデルが応用固有の意味構成要素を組込めるような拡張性を実現する必要がある。

3. **プログラミング言語とのインターフェイス**：データベースを用いてより高度の応用を支援するには、データベース内で種々のプログラム処理を記述できるようにして、より多様なデータベース操作を行なうことが必要である。さらに、これらのプログラムをデータベース内に記憶して部品化や再利用を容易にすることが不可欠である。このためにデータベースシステムとプログラムとの間の呼び出し機構やデータの変換／転送機能が必要となる。

4. **定義情報のデータベース化とその操作**：先に述べたデータ型の定義やプログラムの記述、その呼び出しのインターフェイスといった、様々なデータベース定義のための情報をデータベース管理しておく必要がある。また、これらに対する問い合わせなどデータベースシステムの機能を利用者に提供することも重要である。

このような要求に対して、オブジェクト指向の概念に基いて、(図 2) のような機能を持つデータベース言語を実現している。

これは、主として次のような要素からなっている。



(図3 言語処理系の利用例)

1. **オブジェクト型定義機能は、格納するデータ型を定義し、データベースを表現するためのもので、これにより定義されたデータベースに対して検索や更新、共有制御といったデータベース管理の機能を提供する。**

2. **言語インターフェイスは、データベースシステムとデータベースを利用する応用を記述するための種々のプログラミング言語とのインターフェイス群であり、相互にデータや制御のやりとりを可能にする。型定義機能は処理を記述できないので、そのためにSQLやC等の言語を用いている。**

3. **定義情報データベースは定義された型に関する情報や、内部的な格納形式、データの個数といった情報のほか、メソッドとして書かれたプログラムの名前、パラメータなど呼び出しのインターフェイス等を記憶しており、利用者の理解を助けると共に、この更新によってシステムの特性を変更、改良できる。**

なお、定義情報の操作やad-hoc的なデータベース操作のために、システムの機能のうちSQLによるデータ操作と、型定義の部分を合わせてオブジェクト指向SQLとして対話的なインターフェイスにより提供している。利用例を(図3)に示す。

オブジェクト指向のデータベース環境としては、Smalltalkやc++等のような、オブジェクト指向の言語に基づき、プログラミングも含めて統一的な環境を利用者に提供するといった方向が知られている[7,8,9]が、ここでは**複数のプログラミング言語を混合して使用することを考慮している。**従って、オブジェクトとしてのデータベース定義機能は、データベースの外部仕様を定義するために用いられ、メソッドの内容などを多様な言語を用いて記述できるようにしている。これは、利用者の応用の記述に対して適した言語を使い分けできることを目標としている。

複数のプログラミング言語で書かれたプログラムは呼び出し関係などのインターフェイスに伴う様々な問題が考えられる。ここでは、オブジェクト型を扱い、型検査を行なうことで、プログラム間のインターフェイスにおけるエラー等をできるだけ避けるよう考慮されている。

本稿ではそのうちSQLに基づいた対話型データベース検索言語の実現について述べる。SQLに基づいたインターフェイスを提供することは、次のような特徴がある。

1. **対話的な検索の支援** データベース応用では、アドホック的にデータベースの内容を検索する場合が多い。既存のプログラム言語に比べて、SQLのようなデータベース言語はこのようなアドホック検索には起動が単純で使いやすいと考えられる。

2. **データベース処理記述の容易さ** 条件による非手続き的なデータベース検索や、仮想的な視野であるビューの定義など、データベース言語を用いることでアプリケーションの記述が単純になる場合がある。

先に述べたように、多くのプログラミング言語インターフェイスを特徴に応じて使い分けるような考えに基づくと、データベース応用に対する一般のプログラミングには他のプログラミング言語を用い、アドホック的な検索にはSQL的な対話的な検索言語を提供できるようなデータベース環境を提供することは必要であると考えられる。

4 SQLインターフェイスの機能

先のような考えからSQLに基づいたオブジェクト指向のSQLインターフェイスについて述べる。ここでは、SQLのような言語に対してオブジェクト指向の考え方と、導入するため、次のようなアプローチをとっている。

1. **データベース定義等、オブジェクト指向の概念の導入に伴う拡張** SQLにおけるデータベース定義をできるだけ自然に拡張することが必要である。また、ビュー等データベース言語に特有の機能をできるだけ支援することも重要である。

2. **他言語とのインターフェイスの実現** SQLはメソッドなどの動作の記述には不十分な点が多い。また、他の言語で書かれた利用者のデータベース・プログラムとも相互に利用できるようなインター

フェイスを持つ必要がある。この場合、システム側から利用者プログラムの内部までは検査できないので、インターフェイスはできるだけ安全な方法によることが望ましい。

関係言語に基づいたオブジェクト指向のデータベース言語を考える場合、関係言語では関係の属性などが利用者に開放されているので、このままオブジェクト指向データベースに拡張することは encapsulation の点では問題がある。ここでは、データベースに対するメソッドプログラムの起動に関しては定義されたインターフェイスのみによってアクセスできるが、各属性の値は利用者に見え（この点では encapsulation を満たしていない）ような構成をとり、問い合わせ表現を簡単にしている。

以上のような考えに基づいて、オブジェクト指向SQLにおける、データ表現、データ操作、データ制約の基本的な記述方法を説明する。これによって利用者はオブジェクトとして固有のデータ表現／操作を定義することができる。

4.1 データベースの定義

オブジェクト型の定義は次のように行なわれる。

- **基本型の定義**

```
define type PHOTO(/* データ型の名前 */
        properties /* 型を構成するフィールド */
          (NAME string,
          BODY Image,
          AGE int,
          SIZE int)
        operations /* メソッドプログラムの宣言 */
          (DisplayPhoto((windowNo int)
             returns(int))
          Reverse(.....))
        constraints /* 意味的制約の宣言 */
          (CheckAge(.......)))
```

オブジェクト指向データベース [1,10] では様々な形式の定義が提案されているが、ここで示す表現には次のような特徴がある。

1. 型に適用できるメソッドプログラムは、その名前と入力のパラメータと、出力型といった呼び出しのためのインターフェイスの仕様のみを与えている。プログラムの実現は別に記述されることで、データベース処理を様々なプログラミング言語で記述することを可能にしている。
2. データベースで成立する意味制約を実現するため、constraints といった制約保持のためのメソッドの定義機能を持つ。これは更新時など制約が破られる可能性のあるアクセスの時点で実行される検査プログラムである。

- **継承型の支援**

型定義は他の型から継承することが可能である。このために inherits（親クラス）を持ち、明示された親クラスから型定義情報とメソッドプログラムを引継ぐことができる。

```
define type STUDENT
        (inherits (PERSON)
        (properties(StudentNo int)
                (Grade int))
```

- **ビュー、計算関係の実現**

次の例 (GoodStudent) のように、ビューや計算関係 (computed relation) といったデータベースの各値がアクセス時に作られるようなデータを定義することができる。

```
define view GoodStudent
as select * from STUDENT where Grade >50
```

4.2 メソッドの記述とその組み込みインターフェイス

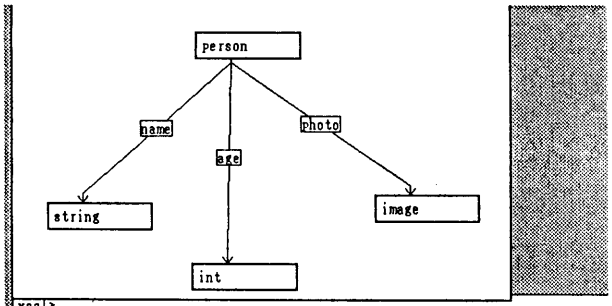
型定義で定義されたメソッドプログラムの実現とその組み込みは次のように行なう。本来は実現する言語を規定しないが、現在は C と SQL を支援している。データ操作は対応する処理を問合せ言語あるいはプログラム言語を用いて記述する。また、データベースの管理外にあるプログラムをファイル名で指定し、呼出すことも可能となっている。

```
define method Child:AVG-YEAR(year int)
returns(int)
as select Person:AVG() from Person
where Person.age < year

define method Image:DISPLAY(windowNo int)
returns(int)
file('/usr/db/method/bin/displayimage',o)
```

上の例で、DISPLAY は Image 型に対するメソッドプログラムで、int 型の windowNo を引数とし、int 型データを返すコンパイルされたプログラム (.o) であり，この場合ファイル名により指定され、データベースに登録されている。また、constraints 句で定義される意味的制約はメソッドと同様な形式で条件または検査プログラムの形で記述する。一般のデータ操作との違いは、問合せに対して自動的にこれらの制約プログラムが問合せ変換法により付加されて実行されることである。

```
define method Child:Checkage()
returns(Child)
as select * where Child.age < 20
```

SQL で記述されたメソッドプログラムはビューや入れ子処理と同様にテキストとして定義情報データベースに格納されている。問い合わせの実行時には、この定義文と、入力された問い合わせを合成して評価することによって制約が実現される（問い合わせ変換法の利用）。定義情報自身も更新の対象となるので、メソッドに関するデータベース検索などが可能である。C プログラムの場合には、一般に C の関数の形で記述、コンパイルされたファイルを実行時にダイナミックリンクを使って呼び出している。システムは型やメソッドの定義時に C の構造体を宣言したヘッダファイル (.h) を生成し、利用者は



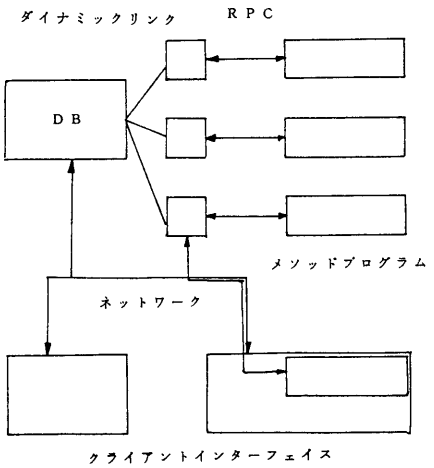

```
xsql>

xsql>define type image (properties(body string,name char[10]) operations(disp
eturns(int)), disp2((p int) returns(STRING))));

xsql>define type person (properties(name string,age  int,photo image) operati
((p int) returns(int)), dispimage((p int) returns(STRING))));

et[pro:/dev/ttyp]
struct  string_cell     {
        struct  string_cell     *next;
        char    *string;
};
struct  image   {
        char    *body;
        char    name[10];
};
struct  s_person        {
        char    *name;
        int     age;
        struct  image   photo;
}       person;

type_to_data(string)
        struct  string_cell     *string;
{
        int     p;
        person.name = string->string;
        string = string->next;
        person.age = atoi(string->string);
        string = string->next;
        image.photo.body = string->string;
--More--(61%)
```

（図 4　データベース定義文と作られるヘッダファイル）

このファイルを include したプログラムを記述、コンパイルする。これは define method 文によってシステムに登録される。メソッド実行時にはデータベースの内部表現からプログラムへデータを渡すために C の構造へのデータ変換をする必要があり、システムが問い合わせの実行時に行なっている。データ定義とそれに伴うヘッダファイルの構成例を (図 4) に示す。

4.3　RPC[13] によるメソッドの記述

先の場合で、利用者の記述した C によるプログラムはデータベースシステムから動的にリンクされ、データベース処理と組み合わせられて実行されている。この場合、入出力のデータ型やプログラム名などは定義情報データベースを用いて検査されるので、データベースに対して不当なプログラムが実行されることは少ない。一方で、言語処理系は利用者のプログラムの内部まで検査できないので、利用者プログラムで誤りが発生した場合にはシステム側で対処の余地が少ない。従って非常に処理時間がかかるプログラムや、誤ったプログラムに制御が渡った場合にはシステム全体の性能などに大きな影響を与える。従って、このような問題のために、SQL インターフェイスでは RPC (Remote Procedure Call) による利用者プログラムの呼び出しを行なっている。この場合、システムは C のヘッダファイルの代わりに、XDR(eXtended Data Representation) と呼ばれる表現形式 (.x) を出力する。利用者はこれを RPCGEN にかけ、結果作られたプログラムと共に利用者プログラムをコンパイルする。言語処理系はそのクライアント部をダイナミックリンクすることで利用者プログラムを呼び出すことができる。この方法には次のような利点がある。



（図 5　RPC によるメソッド起動とネットワーク構成）

1. **タイムアウト等の設定が可能**　利用者プログラムの実行に対して、タイムアウト等の設定が可能になるので、エラーに対するシステムの安全性が向上する。

2. **分散、多言語対応が可能**　RPC は本来ネットワーク上での手続き呼び出しの方法であるので、これによりシステムを容易に分散環境に拡張できる。また、ここで使われている XDR 表現は、機種やプログラミング言語に独立な表現であるから、利用者はデータベースサーバを利用しつつ、多様なローカルな機械でデータベース処理のためのプログラミングを行なえる。

これらは、(図 5) ような実現となっている。

また、利用者インターフェイスの部分も RPC により実現することで、言語のフロントエンドだけをクライアントとして分散して持つことができる。

4.4　集合データ操作の支援

定義、格納されたデータベースに対しては、メソッドによるものの他に、ad-hoc 的な問い合わせを実現するための対話的な集合操作が支援されている。このような問い合わせを支援するのに SQL,Quel といった関係型の集合データ操作を用いることは有益である [11]。ここでは SQL データ操作言語に基づき、次のような機能を拡張して実現している。

1. **部品展開的なデータベースアクセス**
   データ型を部品階層的に構成できるので、その構成要素をたどりつつアクセスできるよう、ドットを使った指定が可能になっている。

   ```
   select from TypeA
   where TypeA.FieldA.FieldC = 'Constant'
   ```

2. **利用者プログラムの呼び出し**
   先に定義したプログラムを次のように SQL から組み込み関数的に利用することもできる。

```
select TypeA:DISPLAY(window1)
where TypeA:Area(*) > 40
```

このように、利用者のプログラムの実行結果(Area(*))を問い合わせの条件として用いたり、問い合わせ結果を専用の表示プログラム(DISPLAY())によって出力することもできる。システムはこれらのプログラムを動的に呼び出して実行している。各プログラムの呼び出しには、対応する型名(TypeA)を指定しており、define-method文内の型の記述と比較することで実行前の型検査を可能としている。

その他には、SQLに基づいた挿入、削除や型データの形式の変更などの機能がある。データの挿入は型データが階層を構成するため、階層的なデータ挿入を行なうよう拡張されている。また、型に基づいて作られたデータベースに対して、型定義を変更することができ(Expand文)、対応するデータベース内のデータをそのままに型定義を変更できる。これは、部分的にデータベースを改良する機能であり、現在の実現では型を構成する属性要素を増やすことができる。なお、新しい属性は空値で埋めるようになっている。

4.5 その他の機能

その他に次のような機能が利用できる。

1. **定義情報データベースとそのアクセス** このように作成されたデータベースの定義に関する情報はそれ自身データベースに記憶され、利用者が用いることができる。例えば、次のような種類のデータが記憶されている。

   (a) 型の定義に関するデータ: 型の名前、属性名、利用できるメソッドの名前、継承型の指定など。

   (b) 型の実現に関するデータ: 記憶ファイルの実現方法、データの個数など。

   (c) メソッドの定義に関するデータ メソッド名、入出力パラメータ、呼び出し型、記憶されているファイルの名前、形式など。

   (d) 制約表現に関するデータ

   利用者の問い合わせ入力にしたがって、これらのデータベースの内容は更新される。システムの処理はこの内容に従うので、これを変更することによってシステムの実現形式を部分的に変更することができる。(例えば、ファイル形式や最適化の手法など)また、このデータベース自身も検索の対象となるので、利用者が必要とする型の名前やその仕様に対して検索問い合わせを出すことができる。

2. **外部インターフェイス** 利用者プログラムから処理系を呼ぶ機能は、現在のところは埋めこみ構文に近い単純な呼びだし形式を持つ。この言語の現在の目的は対話的処理が中心であり、プログラム処理との完全な統合は、ここで実現された内部的なデータベース機能を利用する別の処理系による。利用者インターフェイスはingres等他のデータベースシステムと同様の簡単な対話型インターフェイスを持つ。

5 言語処理系の実現と拡張

5.1 処理系の概要

言語処理系はUNIX上にインタプリタとして開発されている。別に開発されたウインドウインターフェイス[12]と定義情報やデータベースを共有するためなどから部分的に関係データベースシステムingresの機能を利用している。現在の言語処理系の機能としては、HAVING, GROUP BY, ORDER BY といったSQLの機能がいくつか支援されていない。また、VIEW文や入れ子処理も完全ではない。メソッド処理としては、iteratorの機能が支援されておらず、コルーチン的なメソッドは現在では記述が難しい。

データベースはシステムが生成するオブジェクトIDによるハッシュを用いたファイルに格納されている。同じ型のオブジェクトが同じファイルにクラスタリングされている。現在の実現では順アクセスを行なった場合の処理効率と、ハッシュのための記憶効率の低さといった点で問題がある。但し、格納ファイルの指定は定義情報データベースに管理されているので、B-tree等の他のデータ構造にに変更することができる。また、現在の実現では、SQLレベルでは索引を指定する構文を持たないため、オブジェクトとして定義する必要がある。この場合、索引ファイルを利用することができるが、索引を経由した問い合わせアクセスは明示的に索引オブジェクトを記述する必要がある。

集合データ処理の処理方法は比較的大容量の共有メモリの環境を仮定しているので、できるだけメモリ上で処理できるような実行手順を決定するようになっている。問い合わせ最適化の方針は、多くの関係データベースと同様、Selection, Projection といったフィルタを先に行なうといった単純なものである。さらに、定義情報データベース中にデータの個数や大きさ等を記憶しており、処理結果の個数予測により中間結果が少なくなるようファイルのアクセスの順番や処理の相手を選ぶといった、簡単な最適化を行なっている。この時に、実行時に得られた中間結果の個数を使って補正を行ないながら処理できるようになっている点が特色である。

5.2 問題点と拡張

現在の実現では、利用者が記述するプログラムを含んだデータベース処理は、型検査が行なわれるものの、RPCによる処理は完全には実現されていないなので分散処理とシステムの安全性の点で不十分である。また、分散構成に伴う同期制御の手法が組み込まれておらず、少なくともファイルレベルでの同期制御プリミティブの記述、組み込みを行なえるような機構が必要となっている。

6 あとがき

本稿では、LAN環境下でマルチメディア応用の支援を目的とした、拡張可能データベースシステムの構成を示した。また、開発した部分のうち、言語処理系の内容について述べた。これにより、データベースシステムとしての基本的な機能は提供できるようになっている。ファイル構成やオブジェクトの配置、クラスタリングや索引づけなどの高速化の手法を考慮していないため、性能の点では多くの問題があり課題となっている。また、マルチメディア応用の支援には利用者インターフェイスレベルでのデータベース機能の支援が不可欠で、このための機構についてもさまざまな問題が考えられている [14]。

謝辞 本研究の機会を与えて下さる棟上昭男情報アーキテクチャ部長に感謝いたします。また、本研究の開始時より研究全般に対して非常に有益な御議論を頂いた植村俊亮東京農工大教授に深謝致します。ソフトウエアの作成についてはSRA及び富士通SSLに御協力頂きました。なお、本研究は「電子計算機相互運用データベースシステムの研究開発」の一環として行なった。

参考文献

[1] W.Kim and F.Lochovsky,"Object-Orientted Concepts, Databases and Applications", Addison-Wesley, 1989.

[2] M.Brodie (Eds.), "On Knowledge Base Management Systems", Springer-Verlag, 1986.

[3] M.Carey et al."A Data Model and Query Language for EXODUS", SIGMOD"88, 1988.

[4] I.Kojima et al. "The Architecture of an Open Multimedia Database System", 2nd ISIIS symposium, 1988

[5] R.Hull(Eds.) "Database Programming Language " 2nd workshop proceedings. 1989.

[6] 田沼他、「LAN上のマルチメディア分散データベースのための分散記憶システム」情処研報、DBS-66-4,1988

[7] D.Maier and J.Stein, "Development and Implementation of an Object-Oriented DBMS", in Research Directions in Object-Oriented Programming, The Mit Press, 1987.

[8] K.Dittrich(Ed.) "Advances in Object-Oriented Database Systems", Lecture Notes in Comp. Sci. 334, 1988.

[9] W.Kim et al."Features of the ORION Object-Oriented Database System, in [1].

[10] D.Fishman et al."Iris:An Object-Oriented Database Management System", ACM TOOIS, Vo.5, No.1, 1987.

[11] T.Bloom and S.Zdonik,"Issues in the Design of Object-Oriented Programing Languages", OOPSLA "87, 1987.

[12] 小島、植村「拡張できるデータベース設計支援システムについて」信学技報 、1988.

[13] Sun Mycrosystems,"Networiking on the Sun workstation",1988.

[14] 佐藤他、「開放型複合データベースシステムのための、定義可能な利用者環境」、情処全大、3E-8,1988.

[15] M.Stonebraker(Ed.) IEEE Transactions on KDE, special issue on Database Prototype Systems, Vol 2 No.1, 1990.