后关系数据库 DBPL语言 (10)

计算机科学1993 Vol.20 No.6

59-65 TP311.13

后关系数据库语言DBPL

丁宝康 董小浩（复旦大学计算机科学系 上海200433）

摘 要

The "post-relational" database system will become next generation database system. This paper presents the DBPL as a post relational database programming language. We introduce type definition, programming environment, expressions and operations, language completeness, and open system architecture. Finally, by compared with SQL, its advantages in modern database programming are showed.

一、引 言

七十年代处于实验阶段的关系数据库系统在八十年代几乎占据了整个数据库市场，关系数据库SQL语言已成为数据库的通用语言。但是，随着计算机应用领域的扩展，关系模型表现出许多局限性，SQL语言也因其没有充分利用语言技术的命名（naming）、类型（typing）以及联编（binding）等机制，从而没有达到充分的交互性、可扩充性和功能性。九十年代的数据库将突破关系数据库的原有框架，发展成为"后关系数据库"（post-relational database）。本文主要介绍后关系数据库程序设计语言（Database Programming Language，简记为DBPL），这是一个面向集合和谓词的语言，以探索今后的数据库发展方向。

二、从关系数据库到后关系数据库的发展

关系模型作为一种重要的数据模型，为数据库技术作出了三点贡献，但同时仍有不足之处：

1．关系模型具有比较高级的数据模拟能力，可以为各种商业应用提供合理的数据结构，有非常强的查询功能和完整性控制概念，同时也有合适的数据库更新方式。

2．关系数据库技术已经分离和解决了许多难题，例如集合操作、完整性控制等，但是产生了裁剪式的实现，概念独立性问题受到约束：例如只有受限域的记录集才能被定义为永久变量，只在关系的水平结构上提供存取的优化技术，事务的概念有特定、重载的语义。

3．关系模型有一系列各有特色的数据库语言（例如关系代数、元组演算、域演算等），特别是SQL已成为数据库通用语言。但是，尚未达到语言设计的最佳状态，甚至有的阻碍了关系模型的充分利用，导致宿主语言和数据操纵语言的严重失配，例如关系数据库语言的查询操作是一次可以得到一个元组集，但宿主语言往往局限于一次处理一个元组。

因此，许多研究工作致力于研究新的数据库系统，它仍能支持传统的数据库功能，又能超越关系数据库的限制。例如面向对象系统以及各种形式具有"大块数据结构"（bulk data structures）和永久数据（persistence）的程序设计语言。这些数据库可以统称为"后关系数据库"。

收到日期：92-12-10。

与关系数据库相比，后关系数据库具有以下三个特点：

1．关系模型的扩充

（1）引进新的数据类型。数据模型将具有类型完备性和正交性特点；

（2）产生"大的"操作和断言；

（3）提供完整的数据赋值和操作的算法结构。

2．提供各种相互独立的能力

（1）要求扩充的或开放的数据类型和类型构造器的集合具有"持久性"（persistence）；

（2）有效地实现对大块数据访问、模块化、界面定义和控制等的合适的抽象机制；

（3）能支持非功能性的基本机制，例如并发控制、恢复和数据通信等。

3．后关系数据库语言能充分利用现代语言技术根据需要把上述提到的能力捆绑在一起：

（1）能为对象的标识和引用提供合适的命名机制；

（2）提供联编（binding）机制，允许用户选择合适的命名方案、类型、值和可变性约束；

（3）为对象提供灵活的范围、共享和生命期的定义。

从关系数据库到后关系数据库的发展，不仅需要数据模型本身的改变，而且需要语言技术的导向，现在已出现了许多受到语言设计原则重大影响的成果，DBPL是其中的一种。下面将详细考察DBPL各方面特色。

三、DBPL概述

DBPL是一种数据库程序设计语言，1988年提出，随后在VAX/VMS上实现。数据库程序设计语言是为了克服现有数据库中交互性和可扩充性所存在的缺点，为了完善数据库的功能而提出的。它最后将要代替不同的语言界面（查询语言、数据定义语言、报表定义语言等），用单一的语言框架统一不同的抽象层次，它有一致的命名、类型及联编机制，从而有强大的表达能力。

DBPL语言是Pascal/R的后继，为高级数据库程序设计提供统一的语言框架。DBPL设计者的口号是"正交就是力量"。正交的意思是所有的语言概念（如类型、过程抽象等）必须完全不互相依赖，这样在任意的组合时不会相互冲突。正交原则不仅使更多的组合成为可能，也增加了语言的可表达性和可理解性。正交是DBPL的一大特色。

四、DBPL的数据类型定义

DBPL是一种静态的强类型语言，即每个名字在编译时唯一确定地属于某个类型。两个复合对象具有相同的类型的充分必要条件是：这两个对象都用相同的类型名描述。这是一种"名等价"，而不是"结构等价"。

在DBPL中有以下几种类型定义：

1．基本类型（built-in types）。DBPL提供下列基本类型：INTEGER、LONGINT、CARDINAL（自然数）、LONGCARD、BOOLEAN、CHAR、REAL、LONGREAL。这些类型的值可以表示为预定义的标识（如TRUE）或字面值（如3.2E-2，'A'等），也可以是用户自定义的值。例如：

```
TYPE Age=CARDINAL;
CONST Passed=60;
```

2．枚举类型。用户可以使用基本类型定义新的类型，其中枚举类型就是一种。枚举类型可以用名字表定义。例如：

```
TYPE ColorType = (red, yellow, blue);
```

枚举类型中为每个值定义一个序号，使同类型的值能比较大小，例如：red<yellow<blue。

3．子界类型。子界类型可从基本类型或枚举类型中导出。例如：

```
TYPE StudentNum Type=〔890000..
                      990000〕;
     PartColorType=〔yellow..blue〕;
```

在表达式中，子界类型和相应的基本类型的值是相兼容的。

4. 串类型。字符串是由字符序列组成的复合对象。它的类型为字符数组ARRAY[1..maxlength] OF CHAR,其中maxlength的值可以据串类型的不同而不同。例如:

TYPE NameString = ARRAY[0..20] OF CHAR;

5. 数组类型。数组是由固定个数的元素组成，由下标指定数组中的位置，下标必须是枚举型、子界型或布尔型、字符型等基本类型。例如:

TYPE StudentsTable=ARRAY[1..1000] OF StudentType;

其中StudentType可以定义成记录类型。

6. 记录类型：一个记录类型包含有m(m≥1)个成分，用m个不同标识符命名，每个成分有一个数据类型。记录类型的成分称为域(field)。定义记录类型时，需同时指明域名和域的数据类型。例如，记录类型StudentType可以这样定义:

```
TYPE StudentType=RECORD
                   Sno: studentNumType;
                   Name: NameString;
                   Age: [15..30];
                   Sex: (F, M);
                   END;
```

7. 关系类型。关系类型是由同一类型的元素组成的结构，元素的类型称为"关系元素类型"。元素的个数称为关系的基数,不是一个固定值。关系类型需要说明关系元素类型和其键值。例如:

TYPE StudentRelType= RELATION Sno OF StudentType;

这里Sno是键。每个键值唯一地确定(至多)一个关系元素。关系类型的变量可以这样定义:

VAR StudentRel:StudentRelType;

关系StudentRel中的键约束可以这样表达:

ALL s1, s2 IN StudentRel (s1.Sno =s2.Sno) ⟹ (s1=s2)

意思是:关系中任意两个关系元素s1、s2,如果键Sno的值相等，那么这两个关系元素s1和s2相等。

DBPL与传统的关系数据库语言不一样。例如SQL中，一条CREATE TABLE命令把关系类型的定义和关系变量的说明放在一起了。在DBPL中是分开定义和说明的。

从上述DBPL的类型定义可以看出,它充分利用了类型的正交性，通过类型定义，使数据库的表达性大为增强。

五、DBPL的程序设计环境

这里介绍如何应用类型的定义构造模块和应用程序。

1. 模块和应用程序

一个模块由名字定义和语句的序列组成。一个应用程序可以由大量的模块组成。模块可以单独编辑，即独立地开发。一个模块可以有若干个输入名字，这些名字是从包含这些名字的其它模块输出的，或从使用这些名字的模块输出的。

2. 辖域

在模块M中说明的名字n的辖域是在模块M以内以及所有需输入名字n的模块M_i内。在辖域内，名字必须唯一。模块本身也是用一个名字标识的，在DBPL中,模块名的辖域是整个应用程序的范围。

3. 数据库模式的模块设计

现在可以把传统关系数据库系统中的数据库模式也看成一个模块。这个模块是"DATABASE模块"，模块内需说明和输出各种类型的名字、关系类型的变量等。

例1 有一个数据库模式，关于学生和成绩的信息:

S (SNO, NAME, AGE, SEX)
G (SNO, CNAME, GRADE)

这个模式用DBPL中模块定义，需略作修改。把G中的课程名CNAME和成绩GRADE组合在一起，以重复元组形式出现(即属性值又可以是一个关系)。具体的模块描述如下:

```
DATABASE DEFINITION MODULE
  StudyDB;
TYPE
  SnoType:[890000..990000];
  NameType:ARRAY[0..20] OF CHAR;
  AgeType:[15..30];
  SexType:(F, M);
  CnameType:ARRAY[0..30] OF CHAR;
```

```
GradeType:[0.0..100.0];
StudentType=RECORD
              Sno:SnoType;
              Name:NameType;
              Age:AgeType;
              Sex:SexType;
            END;
GType=RECORD
            Cname:CnameType;
            Grade:GradeType;
      END;
GradeRelType=RELATION  OF GType;
GradeType=RECORD
              Sno:SnoType;
              Grades:GradeRelType;
          END;
StudentRelType=RELATION Sno  OF  Stu-
    dentType;
GradeRelType=RELATION Sno OF Grade-
    Type;
VAR
  StudentRel:StudentRelType;
  GradeRel:GradeRelType;
END StudyDB;
```

4. 永久变量和永久模块

一个应用程序也是一个模块，它从数据库模式中输入名字。在模块之间名字的输入输出必须静态地加以说明，这样，在应用程序运行时名字就不可能发生冲突和混淆了。

在DATABASE模块中说明的变量都是永久变量(例1中的StudentRel和GradeRel)，这样的模块称为永久模块。与其它程序变量相比较，永久变量的生命期超过单个程序的执行期。永久变量的生命期比任何使用它的程序的生命期都长。普通模块和永久模块能允许短暂的数据对象和永久的数据对象之间参照使用。

永久变量也是共享的对象，可被若干程序同时访问。但对永久变量的访问必须是一个事务执行中的一部分。

六、DBPL的表达式和操作

DBPL对每一种类型构造器（记录、数组、关系）都有相应的值构造器，用于产生复合类型的对象。例如：

```
v1:=StudentType{902401,
                'WANG', 20, F};
v2:=StudentRelType{{902415, 'LIU', 18,
      M}, {922416, 'WU', 21, F}};
v3:=GradeType{902401, {{'Maths', 80},
                {'PHYSICS', 95}}};
```

DBPL还有三种类型的值的选择器。对于数组中的元素，可用方括号中的下标值来引用，例如name[5]表示姓名中第5个字符。对于记录中域的引用可以用圆点的方法，例如Student·name。关系中元素的引用可用方括号中加键值来引用，例如StudentRel[902401]。有时三种情况可能合起来用，例如：

```
StudentRel[902401].Name:= 'WANG';
```

DBPL为关系类型提供面向集合的查询表达式。这类表达式有三种：带量词的布尔表达式、选择表达式和构造式表达式。

1. 带量词的布尔表达式

这类表达式的结果是布尔值（TRUE或FALSE），并且可以嵌套。例如：

```
SOME Student IN StudentRel (Stude-
nt.Name='WANG')
```

```
ALL Student IN StudentRel(Student.
Age>17)
```

关系元素的比较操作（=、<>、<=、>=、<、>）是根据键值确定的。例如StudentRel1>=StudentRel2等价于下式：

```
ALL*s1 IN StndentRel1 SOME s2 IN
StudentRel2(s1.key>=s2.key)
```

测试集合间的属于联系也可用布尔表达式表示。例如(thisStudent IN StudentRel)等价于下式：

```
SOME s IN StudentRel(thisStudent.
key=s. key)
```

2. 选择表达式

例如：EACH Student IN StudentRel: Student. Sex=F 表示在关系变量StudentRel中选取"性别为女"的关系元素。

在关系类型中也可带有选择表达式指出关系中被选取的元组：

```
StudentRelType{EACH  Studen: IN
StutenRel:Student. Age>20}
```

3. 构造式表达式

这是在其它关系的基础上构造新的关系。例如有一个记录类型的定义：

```
TYPE SGRecType=RECORD
                 Sno:SnoType;
                 Name:NameType;
                 Grades:GradeRelType;
                 END;
```

记录类型SGRecType的值来自StudentRel和GradeRel，可用下面的构造式表达式表示：

```
SGRecType{p.Sno, p.Name, q.Grades} OF
    EACH p IN StudentRel,
      EACH q IN GradeRel:(p.Sno=q.Sno)
```

如果又定义一个关系类型：

```
TYPE SGRelType=RELATION Sno OF SGRecType;
```

那么关系类型的值也可用下面的构造式表达式表示：

```
SGRelType{
  SGRecType{p.Sno, p.Name, q.Grades} OF
    EACH p IN StudentRel,
      EACH q IN GradeRel:(p.Sno=q.Sno)}
```

DBPL除了提供上述表达式外，还提供用于关系更新的集合运算符（:=,:+,:-,:&），这些运算符分别表示关系赋值、插入、删除和修改操作。例如：

```
StudentRel:=StudentRelType{ };
StudentRel:-StudentRelType{EACH s IN
                  StudentRel:s.Sex=F};
```

如上所述，DBPL的表达式的嵌套结构抓住了关系查询语言的实质，用高级的面向集合的选择、构造和更新机制提供了数据抽象。

七、DBPL的三个完备性

从前面几个部分可以看出，DBPL通过统一的命名、类型、联编机制使DBPL偏离了目前的数据库查询语言的轨道。但是把这些特色引入到关系模型后，使关系数据库语言的数据操纵、数据描述、数据抽象等功能显著增强。这个问题从数据库程序设计的三个完备性来阐述。

1. 计算完备性

使DBMS与算法上完备的程序设计结合起来，可以增加数据库语言的表达功能，能对数据库中存贮的数据进行任意复杂的操作。

DBPL吸收了系统程序设计语言Modula-2的数据类型、操作和控制结构，并增加了递归功能、过程以及结构化的语句（例如 IF THEN ELSIF ELSE, CASE, WHILE, REPEAT,FOR,LOOP EXIT等）。因此DBPL是实现复杂数据库应用程序的比较理想的环境。

数据库语言和程序设计语言不应该独立而应该协调地发展，甚至合并起来。DBPL就是把这两方面的概念正交地结合了起来。例如：

（1）关系类型可以出现在任意的内容之中。可以作为数据库变量的类型，也可以作为过程的局部变量的类型，或值参、变参的类型。

（2）带量词的布尔表达式不仅可出现在查询表达式中，而且也可以在条件或循环的结束条件中出现。

（3）构造的新关系可随意在表达式中出现。

从布尔表达式中使用量词SOME和ALL，可见关系演算的使用比关系代数的使用更加广泛。

在数据库中检索数据可能需要循环。例如把男学生的姓名打印出来，就可用下列语句：

```
FOR EACH s IN StudentRel: s.Sex=
    M DO
    InOut.WriteString (s.Name);
    InOut.WriteLn;
END;
```

2. 类型完备性

DBPL还遵循语言设计的“类型完备性”原则，即DBPL中所有类型构造器（例如关系、记录、变量、数组）在语言中有同等地

位。有可能利用这些类型再去构造其它任意的类型（例如关系的数组，或关系中有关系等）。

这些类型的值可以在表达式或赋值式中使用，也可以作为参数使用。DBPL还提供了正交永久性。数据库模块中说明的永久变量不局限于关系类型，这就有可能在数据库模块中去说明永久布尔变量。就像例1所表明的，类型完备性概念很自然地导致关系模型去支持复杂对象的描述，并突破传统关系模型第一范式（属性值不能分解）的限制。

3．抽象机制的完备性

迄今，程序设计语言在大的软件系统中都提供两类重要的抽象机制达到信息的局部化。一种是"处理抽象"，允许程序员从子程序实现中进行抽象，通过引用实际参数表中的名字，去执行复杂的操作。另一种是"类型抽象"，允许程序员从数据结构中进行抽象，然后只要通过自己定义的界面对数据结构进行操作。DBPL通过在语句上抽象的过程，以及在表达式上抽象的功能实现这两类抽象。DBPL还提供选择器在选择表达式上抽象，提供构造器在构造式表达式上抽象。例如选择器：

```
SELECTOR MaleStudents:StudentRelType;
BEGIN
  EACH s IN StndentRel:s. Sex=M
END MaleStudents;
```

这里实际上在关系StudentRel上定义了一个"可更新视图"。

又如构造器：

```
CONSTRUCTOR  SGForMale:SGRelType;
BEGIN
  SGRecType{p. Sno, p. Name, q. Grades}
    OF EACH p IN StudentRel,
       EACH q IN GradeRel:
         (p. Sno=q. Sno)
         AND (p. Sex=M)
END SGForMale;
```

这里实际上定义了一个"不可更新的视图"。

在DBPL中允许递归，如递归查询表达式、递归过程等。这就有可能使DBPL也能表达递归DATALOG程序。

在DBPL中还有一种抽象机制是事务，事务能允许数据库程序员在访问永久的和共享的数据库变量时，从并发和恢复问题中进行抽象。事务相对于数据库而言是一个原子操作。DBPL能保证并发事务以可串行化调度形式执行。例如定义删除操作的事务：

```
TRANSACTION  DeleteStudents (Students:
                StudentRelType):BOOLEAN;
  (*成功时返回TRUE值*)
BEGIN
  IF SOME p IN Students (p. Sex=F) AND
    SOME q IN GradeRel (P. Sno=q. Sno)
  THEN
    RETURN  FALSE;  (*违反引用完整性*)
  ELSE
    StudentRel:-Studets;
    RETURN TRUE;
  END;
END  DeleteStudents;
```

八、DBPL开放的系统结构

DBPL算得上是一个健全的计算完备的自包含语言，但仍需要与外界通讯，例如与用户接口管理系统、网络服务器间的通讯，与其它标准程序设计语言(Pascal、C或COBOL等）编制的软件间的通讯。把数据库程序设计语言建设成为开放的系统结构主要是技术问题，而不是语言的设计问题。为了使在VAX/VMS下运行的DBPL完善成开放的系统结构，在异质多语言环境下做了下列几件事：

1．用一组特别的模块（外定义模块）容纳DBPL编译器辖域以外的过程签名。用这些过程模拟普通DBPL的过程说明。

2．DBPL编译器为每个DBPL模块在标准VAX/VMS连接表中产生一个目标代码文件，这种连接可以把DBPL模块与VAX/VMS编译器产生的目标代码联系起来。

3．其它语言的程序（只要不包含关系类型、选择器和构造器）都能使用个别的

DBPL模块中说明的过程和变量。

4．DBPL编译器产生一个排错表，它由标准的VAX/VMS"多语言排错器"解释。这样就有可能在执行编译过的DBPL程序时设置断点和显示、修改变量。

九、DBPL与SQL的比较

DBPL语言和SQL语言，都是数据库语言，但除了一些简单的功能性差别以外，存在着许多本质的不同：

1．虽然SQL标准中，类型在不断地增加，但是用户不能定义新的类型。而DBPL允许用户定义新的类型，并强调类型的正交性。正交性增加了语言的可表达性和可理解性。SQL没有满足正交的要求。DBPL基本上满足了对一个好的语言设计的最低要求：一种语言，对于它所支持的每个对象，必须提供下列功能和方法：

·构造功能：即从较低的类型中的常量或变量构造类型中某个对象。

·比较类型中两个对象的方法。

·把一个对象的值赋给另一个对象。

·选择功能：从某给定类型的对象中抽取较低类型中成员对象的功能。

·一个通用的、递归定义的语法。

关系数据库系统中对象的相关类型有表（table）、列（columns）、行（row）等，但是SQL无法构造比较或赋值。而在DBPL中，有选择器、构造器和类型构造器、值构造器等，就可进行上述操作。

DBPL不仅具有命名类型的能力，以及重复使用类型的能力，它还是一个强类型语言，可以避免许多不必要的运算错误。它支持类型抽象来隐蔽无关的信息或保护私有信息不受侵犯。DBPL还结合了Modula-2的所有数据类型、操作及控制结构，包括递归函数和过程及结构化的控制语句，所以DBPL是实现复杂的数据库应用程序的理想环境。

2．DBPL的抽象机制不仅包含类型抽象和处理抽象，还提供抽象选择表达式的选择器和抽象构造式表达式的构造器，这两种抽象抓住了关系数据库中可更新视图和不可更新视图的实质。而且选择器可出现在任何关系变量出现的地方，构造器可出现在任何关系表达式需要的地方。

3．DBPL提供类型完备性，即所有的类型构造器享有同等的地位。这样，很自然地导致了支持复杂对象的数据模型和非第一范式关系，从而打破关系模型只限于关系中属性值是基本类型的约束。

4．SQL中所有数据组织成关系（即表格）的形式，简单对象和复杂对象无区别。为了实现共享，共享变量放在公用段中，给数据库实现和维护带来不便。但在DBPL中永久变量即共享变量，用户可以清楚地了解共享变量的说明和使用。另外，DBPL在VAX/VMS连接表上产生目标代码文件，可与所有VAX/VMS编译器的目标代码连接起来。SQL虽然增加了Ada、C等宿主语言，但DBPL可与所有编译器理解的宿主语言耦合。DBPL的开放性比SQL更好。

5．SQL中的关系是用表的形式定义的，一个CREATE TABLE语句产生出一个表，而DBPL可以定义关系类型，并可对其关系元素实行各种操作（:=，:+，:-，:&），加强了"大"数据处理的能力。DBPL还注意正交概念，关系可以嵌套定义，打破了第一范式限制，使应用程序的数据操纵更为灵活。DBPL中的关系是键值与关系元素之间的一个函数，因而自然地完成了SQL缺少的关键字定义的功能。

6．SQL语言与宿主语言是独立的，并且相互之间存在着严重的失配现象。而DBPL把数据库语言与结构程序设计语言有机地结合在一起，就不存在失配问题。（参考文献略）