计算机科学 1994Vol. 21 No. 2

数据库与程序设计语言的紧密集成

孙建伶　何志均
（浙江大学人工智能研究所　杭州 310027）

摘　要　This paper analyzes the "impedance mismatching" between the databases and programming languages, and the various approaches to eliminating it. It's pointed out that Object Orientation is now acting as the catalyst for database/language intergration, since it provides a unified type model for databases and programming languages and other features in which both are commonly interested. Nevertheless the intergration of databases and programming languages is confronted with the culture clash between the two, that leads to many difficulties in designing a database/language integrated system.

一、数据库与程序设计语言之间的阻抗失配

数据库与程序设计语言各自关注的目标不同。程序设计语言主要关注数据结构的定义和操作以及程序控制流程，本身缺乏持久数据的管理功能。对于超过进程生存期的持久数据只有借助于外部低级的文件 I/O 系统。数据库主要关注持久数据管理和共享，本身没有丰富的操作手段，是计算不完备的。对于少量的复杂计算任务，数据库通过调用外部程序设计语言的目标模块完成。数据库与语言模块的数据交换则通过文件进行，存在数据格式的转换问题。

在数据密集型的复杂应用领域，需要将程序设计语言的功能和数据库的功能结合起来，而且通常是以程序设计语言处于支配地位、数据库处于从属地位的方式结合。原因有二：(1)数据是系统生成的，是附属于应用的，系统上层的复杂控制只能由程序设计语言完成。(2)即使对数据库进行扩展使其成为计算完备的，数据库也是资源不完备的[1]，即它无法访问应用程序的所有变量，无法调用必要的系统过程，如数据库不能在应用系统的窗口中显示图形或接受用户输入。

数据库在应用程序中处于从属地位决定了目前关系数据库的 SQL 通常要嵌入到宿主语言之中使用。

然而，数据库与程序设计语言的嵌入式接口方式是极不自然的，存在着所谓的"impedance mismatch（阻抗失配）"的现象[2]。阻抗失配问题体现在两个方面，即程序设计风范的失配和类型系统的失配。具体表现有以下五点：

1. 数据库的数据模型与程序设计语言的类型系统之间存在着不匹配，使得在数据库数据与程序设计语言数据之间有大量转换工作。比如，关系数据库只有平面的关系结构，无法直接表示语言的复杂数据结构；又比如，一般程序设计语言（如 C，FORTRAN，COBOL）都没有提供集合类型，其后果是 SQL 查询所得的元组集合无法由程序设计语言直接操纵，只能借助于 SQL 的指示器(cursor)，一次返回一个元组给语言程序处理。诸如此类的数据转换工作在数据密集型应用中占整个应用代码的比例通常达 30%[3]，甚至更多。这不仅对开发人员是一个沉重的负担，而且增加了系统模块之间的接口，是对软件模块性的严重破坏。

2. 由于没有跨越语言程序和 DML 代码的类型系统，限制了结合部的类型检查，代表数据库对象结构的程序变量可能被非法赋值，而这种错误要推迟到数据库事务提交时才能检测得到（如果能检测得到的话）。

3. 查询只能对持久对象（比如，关系）进行，无法对易变对象进行，也不能对调入易变存储空间（内存）的持久对象进行查询，这是一种查询语义的不一致。

4. DML 和程序设计语言语句不能自由组合，如，不能对函数返回的集合类型结果进行查询，也不能以查询结果直接作为函数调用的集合类型参数。

5. 两种语言的语法和语义完全不同，应用开发

孙建伶　博士，主攻数据库、人工智能、软件工程。何志均　教授、博士生导师。

人员必须学会两种语言，体会两者之间的区别，这对开发人员是一个负担。

数据库和语言之间存在的阻抗失配现象对高级数据库应用是一个极大障碍。它破坏了系统的模块化，降低了系统的性能；增加了应用开发人员的负担，降低了软件生产率。

二、解决阻抗失配问题的途径

为了摆脱数据库与程序设计语言之间的阻抗失配所造成的困扰，人们作出了各种努力，有的是权宜之计，有的则致力于问题的根本解决。

1. 非根本途径

(1)特制数据管理系统。关系数据库的低效性连同与程序设计语言之间的阻抗失配使得一些数据密集型复杂应用系统(如 CAD)索性放弃采用商品化关系数据库管理系统，而开发自己系统特定的工程数据管理系统，如 CAD 软件 Euclid、CV、Intergraph 等，都是这种思路。这种做法的缺点是通用性差，功能不完备，开发代价相对太高。

(2)查询抵载(query download)。为了避免频繁与数据库系统交互，一个显然的办法是利用查询一开始就将应用程序要用到或将要用到的数据从数据库中取出来，把这些数据转换成程序语言数据结构，用好后再作相反方向的转换。这种办法还使得程序设计语言可以在内存中对数据作快速处理。缺点是起动和完成时应用程序过载；而且 DBMS 强大的功能不能有效利用，并发访问受到了限制。

(3)二进制大对象块(BLOB)。在 BLOB 作为基本类型之一的数据库系统中，应用程序把复杂数据结构打包成 BLOB，作为数据库中的 BLOB 属性存入数据库中。BLOB 可以在数据库中高效存取，因为只需一次数据库调用。但是采用这种方法使数据库的大部分功能都无法利用，因为 DBMS 无法意识 BLOB 的内部结构，使得查询、并发控制和其它一些操作无法发挥作用。

(4)过程式数据库语言。如果数据库查询语言具有一般程序设计语言的操作和控制结构(如 SYBASE)，或者从查询语言中可以调用程序设计语言的过程(如 INGRES、POSTGRES、Starburst)，那么应用程序全部或大部分都可由数据库查询语言编写。但是，即使这样，这两种 SQL 却又是资源不完备的，对于复杂应用系统不合适。

2. 数据库程序设计语言

以 Pascal/R 为先导的数据库程序设计语言，其基本思路是将数据库模型(通常是关系模型)紧密地结合进传统程序设计语言中，并将相应的关系操作和控制结构也引入语言之中。属于这一范畴的语言有 Astral、Rigel、Theseus、Plain、Modula/R 及 RAPP[4]。

在 Pascal/R 中引入了 RELATION 类型作为与关系数据库模型中关系的对应；DATABASE 类型则作为构造数据库的手段。定义在 DATABASE 之内的 RELATION 类型变量被认为是持久数据；定义在 DATABASE 之外的 RELATION 类型变量被认为是易变数据，这类变量用来存放程序关系运算的中间结果。

Pascal/R 中，RELATION 类型的变量完整地和语言结合在一起，它们不仅可以是永久数据库的一部分，而且在程序中可以被说明为局部变量。此外，在程序中关系变量既可以当作数值，也可以作为参数。

Pascal/R 提供以元组关系演算为基础的存取和处理关系的专用操作，并提供存在量词和全称量词，以此可形成复杂的查询表达式。Pascal/R 还提供 RELATION 类型变量的赋值、插入、删除、替换等操作。

Pascal/R 的缺陷是不满足类型完整性[4]，因为只有 RELATION 类型的对象才能持久化于 DATABASE 中；而且 RELATION 类型是平面结构，它的属性只能是 Pascal 的简单类型，不能是枚举、集合、数组、记录等高级类型。

Pascal/R 还有其它缺陷，如，程序中只允许有一个数据库，整个程序的一次运行作为一个事务等。

3. 持久性程序设计语言

持久性程序设计语言采用了更加精致的持久性模型，在该模型中任何数值都可以持久化，与类型无关；持久对象作为程序变量须经相同的严格类型检验。语言 PS-Algol[5]是首先以如此一致的方式对待持久性问题的语言。

与 Pascal/R 的思路不同，PS-Algol 在语言中不引入新的数据结构，而是把持久性作为提供给现有数据结构的一个性质。PS-Algol 的持久性有两个特点，其一是持久性与数据类型的无关性；其二是数据类型的完整性，即所有数据对象的操作规则是一致的。

PS-Algol 有一种通用指针类型 Pntr，它是非类型化的，不被约束为一种特定类型。因此程序员在利用 Pntr 定义记录类时必须使用注释来说明其含义。利用 Pntr 可以定义多态函数，这是 Pntr 的有用之处。尽管 Pntr 是非类型化的，但 PS-Algol 仍然被认为是强类型的，这是因为通过 Pntr 访问结构字段时会引起动态类型检查。

PS-Algol 提供了打开数据库、事务提交及数据库

入口管理等函数，PS-Algol 还可以把过程作为数据库的一部分存储。并能象操作其它程序对象一样操作过程。

PS-Algol 以及其它持久性程序设计语言，通常不支持任何形式化数据模型，也不支持相联查询（只提供导航式访问）。此外，在当前的实现中，这些语言中没有一个支持事务处理和恢复。尽管如此，持久性程序设计语言所提供的持久性模型为数据库与程序设计语言的结合打下了坚实的基础。

数据库程序设计语言与持久性程序设计语言这两个领域都有多年的历史，目前正朝着统一的方向发展，面向对象数据库刚好处于两者的交叉点上。

三、面向对象与数据库/语言的集成

1. 面向对象作为数据库/语言集成的催化剂

数据库和程序设计语言之间阻抗失配的一个主要表现是两者类型系统之间的不匹配，面向对象为两者提供了统一的类型系统，而且是两者都向往和乐于接受的类型系统。

除了类型系统外，面向对象还为数据库和程序设计语言提供了双方共同感兴趣的东西，如模块性、可维护性、软件重用等。所以面向对象促进了数据库与程序设计语言的集成，起到了数据库/语言集成的催化剂作用[5]。面向对象数据库系统（也称为面向对象的程序设计语言）即是面向对象程序设计语言与数据库结合的产物。

2. 面向对象数据库程序设计语言的持久性模型

将持久性融入面向对象的程序设计语言中，需遵循下列原则：

（1）持久无关性原则：持久性是对象实例的属性，而非对象类型（或类）的属性，即持久性与类型无关。这意味着不必为结构和行为完全相同的持久对象与易变对象定义两套类型。

（2）类型完整性原则：任何类型的对象，包括任意复杂的用户定义类型的对象，都可以具有持久性。

（3）操作一致性原则：持久对象和易变对象的访问和操作方式完全一样，这意味着，程序代码与被访问和操作对象的持久性无关。例如，假如我们有以对象类型 T 做为参数的函数 f，那么 T 的持久和易变对象实例都可以做为实参调用 f。

持久性三原则的好处是显然的，即编程人员以一致的观点看待和处理类及对象，持久性的支持对用户来说是透明的，用户不需为持久性付出额外的代价。

上述持久性的三个原则，其重要程度不是一样的，而且目前商品化的 OODBMS 也并非遵循全部三原则。大多数 OODBMS 实现了持久无关性原则，如 Zeitgeist、VERSANT、ObjectStore。ObjectStore 还实现了类型无关性原则和操作一致性原则。VERSANT、ONTOS 的持久类必须从系统定义类继承下来（分别是 POject 和 Object），所以没有实现严格意义上的类型无关性原则。VERSANT 对持久对象写访问之前必须先调用方法 dirty，所以不遵从操作一致性原则。

四、两种文化传统的冲突

虽然面向对象为数据库和程序设计语言提供了统一的类型系统及双方共同感兴趣的东西，但这并不意味着数据库与语言的紧密集成已经没有障碍了。事实上，数据库与程序设计语言各自有深刻的文化传统，有自己的假设和基本原理。双方的许多特征在各自看来都是非常好的，但是放在一起时，就会产生不相容性，甚至产生激烈的冲突[6]。

数据库最初是从商业数据处理应用领域发展起来的，这些应用需要大规模数据的持久和共享。为了共享，数据被看作是程序外部的。数据独立于各应用程序，而由一个专门的进程（称为 DBMS）所控制；各应用程序通过一个中性（与特定程序设计语言无关）的数据操纵语言 SQL 访问数据库；DBMS 负责权限控制、查询处理、并发控制等；数据库先于应用程序而设计，数据库管理员负责均衡相关各应用的数据视图和需求，产生全局概念模型，又分别映射到各用户视图；数据模式的改变不影响应用程序。由于程序与数据独立、应用的语义在应用程序中实现，DBMS 只知道数据模型的语义。所以，为了更好地反映应用，数据库研究试图不断地为数据模型增加语义，如约束机制、触发机制、导出数据、逆关系等等，但这些机制却并不容易加到语言之中。

程序设计语言（这里指用于大型软件开发的编译型程序设计语言，如 C、C++等，而不是指 Prolog、Lisp 等人工智能语言）注重复杂有效的处理功能（而不关注数据共享和持久性）。为了效率，程序通常编译执行，因此并不考虑类型的动态进化；复杂结构的数据是局限于程序的，对数据的访问通过指针追踪方式（而不是相联查询方式）。由于不强调共享，所以程序设计语言只有文件级权限（而不象数据库一样有粒度很细的权限控制）。程序设计语言讲究软件的可靠性和可维护性及相关的抽象和模块化机制。对语言要素的选择十分讲究，不仅强调通用性和有效性，而且关注各要素间的交互作用，各语言机制之间

复杂的或难以预料的交互作用是不希望的。另外，语言机制的易于理解和使用也是十分重要的。

数据库与语言的集成目前碰到的最大问题是功能选择问题，即哪些数据库特征和语言特征应该集成在一起。因为并不是所有的数据库、语言特征都能结合在一起，许多数据库特征若加入到语言中会破坏语言的基本目标，即抽象、模块性和有效性。

比如类(Class)的概念，数据库与程序设计语言的看法是不一样的。数据库把类看作同一类对象的集合，系统自动维护对象的增加和删除；对象集合作为通过相联查询访问对象的入口。程序设计语言则趋向于把 Class 看作类型(Type)，作为同一类对象结构与操作的定义。程序设计语言认为把类作为集合是不合适的，因为把可能完全不相干的同一类的不同实例聚合在一起，使得应用程序潜在可以访问所有实例，这是违反抽象原理的。另外，象废料收集(C++标准化的下一个目标)这样高级的存储管理功能与把类作为同类对象集合是相冲突的，因为每个对象存在于其所属的集合(类外延)中，都可能通过查询被访问，使废料收集无法进行。

总之，数据库与程序设计语言的许多特征集成在一起时会产生冲突，这种冲突主要倒不是技术上的，而是"观念"上的冲突，是一种"文化传统"冲突，冲突迫使 OODB 设计人员在许多特征中作出选择[7]，以便特征的集成能较好地满足双方的需要。事实上，目前商品化 OODBMS 或原型系统都是这种冲突与权衡的结果，有的比较偏向程序设计语言传统(如 ObjectStore)，而且目前这种情况占主流。这可能是面向对象思想在程序设计语言中首先得到实践的缘故。不过也有人认为，OODB 今后可能会转为偏向数据库传统为主[8]。

五、小结

数据库和程序设计语言之间存在着风范和类型系统之间的失配，这种失配是软件模块性差，可维护性差和可靠性差的一个重要原因，也是软件生产率的障碍。为消除这种失配，人们做了大量努力。面向对象为程序设计语言和数据库提供了统一的类型系统及其双方共同感兴趣的东西，成为数据库/语言集成的催化剂。面向对象数据库正是以面向对象集成数据库和语言的产物。但是数据库与语言并不是类型系统统一就万事大吉了，数据库文化传统与程序设计文化传统存在着冲突，这种冲突使得双方的许多特征无法方便地集成在一起。面向对象数据库程序设计语言要取得成功的关键要素之一是功能选择，即如何权衡、选择数据库和程序设计语言的功能特征集成在统一的环境之中，以满足数据库和程序设计语言习惯者双方的需要。

(参考文献共 8 篇略)

下期主要内容预告

交互作用之基础
按自然法则计算导论
日本新一代计算机计划之回顾
开拓新时代
FGCS 计划十年总观
高级数据库系统基于语义的并发控制
工程数据库模式设计的一种形式方法——尼杰森法
机械智能 CAD 的结构和发展趋势