对象关系型数据库的体系结构

Donald D. Chamberlin （美国计算机学会）

本文将通过一个实例详细考察对象关系型数据库系统的对象体系结构，这个实例就是刚刚面世的 IBM DB2 通用数据库服务器。

1.**大型对象**

现代数据库应用存储对象的最重要特性之一是其绝对尺寸，对于非结构信息如图象、声音和视频更是如此，例如1分钟压缩视频依据其压缩方法的不同需要10MB至30MB的存储空间。如果用常规方法处理，视频剪辑和其他大型对象将会占用了几乎所有资源，包括缓冲区和日志记录，从而会大大降低系统性能。虽然许多数据库系统都可以存储大型对象，但一个真正的对象关系型系统必须提供特殊的方法，以提高大型对象应用的性能和尽量减少大型对象对系统资源的冲击。

DB2 提供了三种数据类型存储大型对象：BLOB 用于存储二进制对象，CLOB 用于存储字符串对象，DBCLOB 用于存储双字节字符串对象，这三种 LOB 类型都能够存储最大达 2GB 的对象。当 LOB 数据存储在一张表的某一列中时，对每一个 LOB 值该列中将包含一个“描述符”，大型对象本身则存储在表格以外的地方。这种存储策略保护 LOB 值不干扰表格物理集聚，但可能增加扫描表时的引出页数。CREATE TABLE 语句选项使表格创建者能够控制 LOB 型列在物理介质中的放置。

DB2 恢复子系统包含对大型对象的特殊处理，对于每一个 LOB 类型列，表创建者可以选择关闭日志功能以保存日志空间。如果对一个列关闭了日志功能，仍然能够保证事物处理的一致性，但这一列不能参与前滚恢复（在数据库从一个存储介质检查点恢复之后重做先前的事物处理）。

由于 LOB 的尺寸，人们非常希望能够最小化移动或复制 LOB 值的次数。出于这个原因，DB2 容许应用程序声明一个“定位器”变量来代表一个 LOB 值，而实际上并不包含 LOB 值。定位器的内容是一个“处方”，说明必要时如何得到 LOB 具体值，一个定位器可以在任何 SQL 表达中代表一个 LOB 值。对定位器的操作非常有效，因为这些操作是通过操作“处方”而不是实际 LOB 值来实现的。例如，如果 LOC1 和 LOC2 是定位器变量，表达式 LOC1||LOC2 只是把这些定位器变量所包含的处方连接起来，而不是实际连接定位器变量所代表的 LOB 值。通过使用定位器，应用程序可以完成对 LOB 值的一系列操作，从而把实际 LOB 数值的具体化推迟到最后一步。

用户经常需要从一个文件向数据库引进一个大型对象，或从数据库输出一个大型对象到文件中。DB2 允许应用程序在数据库和文件之间交换 LOB 类型数值，不需要通过程序缓冲区移动数据。一个程序可以声明一个“文件引用”变量，包含特定文件的名字，随后文件引用可以作为代表大型对象文件内容的输入或输出变量用在 SQL 语句中。通过使用定位器和文件引用，应用软件就可以经常处理大型对象，而无需把实际对象取到程序存储器中。

2.**用户自定义函数**

DB2 提供了 100 多个内置函数以执行对数字、串、日期和其他类型数据的各种计算，此外它还允许用户用 C、C++ 或 Basic 创建他们自己的新函数。用户定义函数（UDFs）可以带参数，能够用于任何希望获得标量值的 SQL 表达中。

DB2 定义了将一个 UDF 参数传递给执行这一函数的用户程序程序的约定，该 UDF 的输入参数连同将要存储函数的结果和状态代码的缓冲区指针一起被传递给执行程序。函数创建者必须编译这个执行程序，并在数据库服务器可以使用的目录中安装所得到的可执行文件。

编写和编译执行新函数的程序后，用户必须执行一个 CREATE FUNCTION 语句为这个 UDF 注册，如果这个函数将被用于一个以上的数据库，那么必须在每一个数据库中分别注册。CREATE FUNCTION 语句定义了该函数的参数类型和结果类型，告诉系统到哪里去找执行程序，并将对这个函数的描述添加到系统目录表中。函数注册之后将被动态加载，只要调用到该函数，就立即执行。保护执行函数的可执行文件非常重要，因为在调用函数时，系统不再做进一步检查便立即使用该文

件。

函数的名字和参数类型一起被称作函数的“签名”,像大多数其他现代编程语言一样,SQL可以定义几个名字相同但参数类型不同的函数来“重载”一个函数名字。在处理一个给定的函数调用时,DB2将调用参数类型与调用函数实际变元类型最为匹配的函数。例如,一个保险公司定义了两个计算估计寿命的函数:以生日和性别为变量的LIFE-EXP(DATE, VAR-CHAR(6)),以年龄和职业为变量的LIFE-EXP(INTEGER, VAR-CHAR(32)),调用LIFE-EXP(DATE('1945-06-15','Female')将引用第一个函数,而调用LIFE-EXP(60,"Sky Diver')将引用第二个函数。

3.用户自定义类型

内置的SQL数据类型(例如Decimal和Double)常常用于表示有具体含义的数据,例如,一个Decimal列可能包含美元值,而另一个Decimal列可能包含日元值。数据库设计者头脑里可能有一些对不同数据类型有意义的操作类型规则,例如将两个美元量值相加是有意义的,但把一个美元值与一个日元值相加就是错误的。如果只使用内置数据类型,DB2就无法知道哪一种特殊规则适用于数据,因此,系统可以让用户你创建自己的数据类型,并指定适用于他们的操作。在DB2中,用户定义数据类型被叫做“单值类型”,每一个单值类型与某一个内置类型共享一个公共表示,这个内置类型就被称作该单值类型的“基本类型”,但单值类型有自己一套合法操作。

当用户创建了一个特殊类型时,DB2会产生类型转换函数,进行一个数值的特殊类型与相应基准类型之间的转换。当第一次创建一个特殊类型时,适用于它的唯一算符是比较两个具有同样特殊类型的数值的比较算符。例如,如果SALARY和BONUS是DOLLAS类型数据列,则SALARY = BONUS and SALARY > BONUS是合法的判断,但SALARY + BONUS和SALARY * BONUS是不合法的,因为没有对DOLLARS类型定义算术操作。

很容易说明哪一种基本类型算符对用户自定义的特殊类型有意义,每一个内置算符(如“+”)是通过一个与算符同名的函数来执行的。为了使一算符适用于用户的单值类型,只需要简单地创建一个与算符同名的新函数来接受参数并返回单值类型的结果,该算符函数可以源于IBM提供的内置算符函数,这意味着它的执行依赖于IBM函数。通过创建源函数,用户可以确切地说明一组对于单值类型有意义的算符,然后可以依靠DB2捕捉到查询和应用中任何与类型有关的错误。

当然,除了为基本类型定义的算符之外,用户可能还希望自己单值类型具有一些其他功能,例如可以基于内置类型VARCHAR(50)创建一个ADDRESS的特殊类型,然后可以创建一个用户定义函数TIMEZONE(ADDRESS)计算一个给定地址的时区。与其他UDF一样,TIMEZONE函数可以用C、C++或Basic编写,还必须在将要使用它的数据库中注册。

在对象关系型系统中,捕获存储对象的状态和行为是很重要的。在DB2中,一个单值类型的行为是通过定义在这种类型上的函数捕获到的,例如,unary函数(如TIMEZONE(ADDRESS)和ZIPCODE(ADDRESS))可以被认为是一种定义ADDRESS类型的行为的“方法”,因此,在DB2中,数据本身和数据行为都是可以被多个应用软件共享的资源。

4.活性数据

前文中曾把“活性数据特性”定义为一种机制,由此一个SQL语句可以激发某些在该SQL语句中没有直接说明的操作。活性数据特性对于保护数据完整性、处理异常条件、产生丢失数据和维护数据库变化的审计跟踪非常有用。一般说来,一个活性数据特性使用户能够指定系统要执行的规则类型。在数据库中定义这些规则——而不是在每一个应用中定义——避免了冗余和不一致性,简化了应用开发者的任务。活性数据在客户机/服务器环境中特别重要,因为它提供了一种适用于所有应用的全局规则的定义机制。

DB2的活性数据特性可以分为两类:“约束”和“触发器”。约束是一些系统自动执行的简明性语句,例如“鞋尺寸总是正的”或“每一个雇员有一个经理”;触发器是一些自动操作,当探测到一定的事件或条件时,这些操作就会被自动激活,例如“只剩下10只铅笔时,定购10只新铅笔”或“记录谁更新了ACCOUNTS表”。

DB2支持下述类型的约束:NOT NULL约束,NOT NULL约束,禁止在给定的列中出现空值;UNIQUE约束,防止在给定的一列或几列中包含重复数据;PRIMARY KEY约束声明一列或数列具有UNIQUE和NOT NULL属性;CHECK约束,这是一些判断,例如系统自动地执行BONUS〈SALLARY判断;视图CHECK选项,适用于一个视图,防止通过视图插入或更新与视图定义矛盾的数据;FOREIGN KEY(引用完整性)约束,在两个表之间施加上一种叫做“父表”和“子表”的关系,要求子

表中的每一个非零 FOREIGN KEY 值在父表中有一个匹配值。

一个约束像是一个声明性规则，触发器则更像一个“精灵”，只要特定的事件发生就会被唤醒，并按你的吩咐去做。DB2 触发器支持的部分选项如下：触发器可以被一个给定表插入、删除、更新或一个表中指定的某一列的更新激活；可以指定在一个激发事件之前或之后来执行触发器；当触发器被一个 SQL 语句激活时，该触发器可以再执行一次，也可以对 SQL 语句所修改的每一行（可能是 0、1 或多个行）执行一次；触发器激活时，可以计算一个判断，称作“触发器条件”，只有触发器条件为真时，触发器才继续执行触发器本体；触发器主体可以由一个或多个 SQL 语句组成，在这些语句中，可能会用特定变量表示激活触发器的一行或数行的“新”与“旧”数值。如果触发器主体要修改数据库，则会对触发器的定义者进行授权检查，而不是其 SQL 语句恰巧激活了该触发器的用户。这个过程使触发器定义者能够以有用的形式“封装”某些特权，使特权级别较低的用户也可以使用。

作为一个例子，让我们来看看一个触发器怎样自动地维护一列数据。假设用户的数据中包含一个 STOCKS 表，表里有 SYMBOL、PRICE 和 HIGHPRICE 列，在这张表中存储了不同股票的数据。一种股票的当前价格总是被维护在 PRICE 列中。由于股票的价格一直在波动着，用户会希望把每一种股票的最高价自动地维护在 HIGHPRICE 列中。这一目标可以用下述的触发器来实现。

```
CREATE TRIGGER stockhigh
NO CASCADE BEFORE UPDATE ON stocks
REFERENCING NEW AS newrow
FOR EACH ROW MODE DB2SQL
WHEN (newrow. highprice IS NULL OR newrow. price> newrow. highprice)
SET newrow. highprice = newrow. price
```

约束和触发器者可以执行企业规则，保护数据库的完整性，增加数据的价值。然而每一种活性数据特性有其自己的优点。约束以声明的方式来表示，从而给系统以更多的优化机会。与触发器不同，约束在创建的时候就开始起作用了，如果某些现有的行违反了一个新约束，则该约束就要被弹回或者违反约束的行被移到一个异常表中。同样，一个约束可以使数据库预防无效条件，但不管是否达到无效条件，触发器只会被特定的插入、更新或删除操作激活。

另一方面，触发器比约束功能更强一些，因为触发器可以访问“新”和“旧”的数据值。这一访问使触发器可以执行这样一个规则，如“工资从不减少”，而约束则不能施加这种规则。此外，约束被限制在一定的预定义操作上，而触发器可以执行任何 SQL 语句或顺序串语句，这一能力使触发器可以完成约束所不能完成的操作，例如更新数据库、写到文件中或产生某一个特定错误代码。一般说来，如果可能的话，一个企业规则应该用约束的形式表示，否则可以用触发器。

5. 协同作用

大型对象、用户定义类型和函数、约束和触发器都是功能强大的特性，但 DB2 真正的对象关系型威力来自这些特性一起使用时的协同工作。作为这种协同性的一个例子，我们考虑 DB2 的对象关系型特性怎样被用于在数据库中存储多边形，多边形在图象、计算机辅助设计、城区规划和其他应用领域中有很多用途。

由于“多边形”不是一个预定义数据类型，我们面对的第一个问题是在数据库中如何表示一个多边形。因为一个多边形可能会非常大，我们使用一个大型对象类型（BLOB）来表示它，但我们将把 BLOB 的这一用法与所有其他通过单值类型创建 BLOB 的用法区别开来。下面的语句创建一个最多可包含 1MB 数据的多边形类型：

```
CREATE DISTINCT TYPE POLYGON AS BLOB (1M);
```

在创建了一个单值类型之后，需要为多边形选择一种表达方式，我们希望得到的多边形行为可以通过创建一组操作多边形的 UDFs 来建模，这些函数当中至少有一个应该是“构造”函数，该函数可以从更基本的类型（例如 POINT 或 DOUBLE）来创建一个多边形。这个构造函数把多边形的各个基本部件放到一个 BLOB 中，然后用系统产生的 casting 函数 POLYGON(BLOB) 把这个 BLOB 转换成 POLYGON 类型。

6. 展望

DB2 提供了对象关系型系统的两种基本组件：一个实现面向对象功能性的体系结构和在这一体系结构之上实现的一组关系型扩展器。借助 DB2 的体系结构，用户可以定义自己的新数据类型、函数和规则，以满足企业不断发展的需要。在现代数据库应用中，数据容量和复杂性都在迅速增加，对象关系型数据库系统由于其强大的功能和灵活性非常好地满足了这些要求，通过存储大型复杂对象和捕获数据语义行为，对象关系型系统大大地增加了数据库产品的价值。