面向对象数据库能否象关系数据库一样成功？

Joseph Dawson

八十年代初，当人们澄清了大型数据库所出现的问题时，关系数据模型就成了数据库设计技术的不可缺少模型。但是人们发现，关系模型对于处理某些类型的应用显得力不从心，特别是象CAD和计算机辅助软件工程这样复杂的应用更是如此。例如，一个电子工程师的CAD软件一般包括方案定型（Schemacapture）编辑器、设计规则检查器和线路设计程序，所有子系统都要求有大量的持久性数据。这样的应用要求是传统数据库所不能轻易满足的，这包括为非常复杂的数据建立模型和在不影响当前应用库的条件下扩充数据库的能力。最近几年，研究人员已经开发了能较好地满足复杂应用需要的面向对象数据库管理系统。

面向对象

在面向对象的程序设计环境中，一个对象是一个具有私有存储和公共界面的实体。人们可用消息指示对象报告或修改私有存储的内容。消息是由过程（即方法）实现的，在访问对象的私有存储方面，这些过程有其特定的权限。所有对象都属于某个类（Class，即类型type），类定义了对象能识别并作出响应的消息。类继承其超类里的所有消息。简言之，一个对象由私有数据和能对这些数据进行操作的方法组成。

面向对象数据库和面向对象语言源于同一概念，但面向对象数据库又增加了一些特征，如持久性、并发控制、可恢复性、一致性和查询数据库的能力。可以用一个计算上完整的编程语言加上数据库本身的许多例程对面向对象数据库进行程序设计。由于数据库中含有许多应用例程代码（这是共享的所在方面），这些程序有可能共享现在代码中的应用语义。数据库系统可利用这些程序所补充的知识来优化查询处理和控制事务的并发执行。

和关系数据模型不同，已经出现了单一面向对象的数据模型，而且继续在对共享若干高级特征的一些模型进行研究。少量商用产品也正在进行开发。

尽管没有统一的数据模型，但对面向对象数据库设计的研究仍有许多共同的目标。其中目标之一是为进行扩充提供工具系统。由于新的应用常常包含难以预料的复杂数据形式且随时发生变化，因此需要可扩充性。一套固定不变的数据构造原件难以充分地支持任何新设计的数据。对数据模型扩充附加功能所达到的程度应与内定义原件无区别。进行扩充的一种办法是创建新类型。类型是指示如何操作该类型实例的样板。在编程语言中，通常实施类型检查以确保表达式的类型与使用它们的上下文一致。例如，进行赋值时，操作符左侧变量的类型必须与右侧表达式的类型兼容。

类型系统的一个重要方面是，类型检查是在编译时还是在运行时进行。数字设备公司开发的Trellis/OWL语言综合了强类型、抽象数据类型和继承性。该语言的类型检查是在编译时完成的。

创建新类型对于数据库并不新鲜，但类型封装它的表示却是个新思想。早期的数据库模型给用户提供一个固定的内定义类型集合和很小的类型构造器（如记录）集合。用户用类型构造器可以构造新类型，但这些新类型所允许的操作应与类型构造器所定义的那些操作相同（例如，对于记录，其基本的操作是在它的域上取值和设值操作）。换句话说，用户不能隐藏新类型的表示。

封装使得人们能利用模块来建立自己的系统，而对模块的访问得通过精心定义的界面。抽象数据类型方法通过一组强类型操作（或方法）指明符来定义界面。它要求每个类型T定义一个表示R（即某个其它数据类型）。每当创建一个T的实例时就必须为R分配一个实例。这种表示存储了对象的状态，仅允许方法访问这种表示，因此用户改变了它的表示也不会扰乱系统的其它部分——所需做的全部工

作是对方法重编码。

也可以把面向对象的数据模型表征为通过对象的标识进行引用的能力，（标识是当对象修改了它的状态时对对象不变的称呼）。可以用这个标识指明一个对象。近来指针已成为大多数现代编程语言的部件，某些早期数据库模型（如CODASYL）也有这种部件。相比之下，关系模型是基于值的，因为它们没有这种标识的概念。

面向对象数据模型的另一个特征是它包含有所谓继承性机制的类型模式。有了继承性，类型定义就可通过一个类型网络与其它类型发生联系。这种基本思想是增量地增加子类型定义以修改原有类型。超类型与子类型的结合可得出一个全新的类型。

数据库的考虑

面向对象的数据库是第一流的新式数据库，正因为如此，它必须提供人们所期望的现代数据库系统所有的特征和功能。这些特征包括持久性、并发控制、恢复能力、一致性以及相联访问。

持久性是对象的寿命比创建该对象的处理过程长多久的能力。一个持久的对象存在于将不依赖于任何单个计算实体的存储空间中。这个持久的存储空间——数据库——能存储大量的对象，不只是配备一个进程的虚存。一般说来，还应提供一些特定的存储结构（如B-树），使用户能对对象的集合进行高效的查找和访问。

多个并发进程（即事务）可共享持久数据的存储空间。共享的媒介通常就是对象。并发访问共享的对象要求对这些事务的操作同步化，使之不会产生意想不到的结果。

如果出现了系统故障（不管是硬件或软件），在防止出现不一致的意义上，数据库必须有恢复能力或容错能力。大多数数据库系统是通过将应用的各项工作划分成多个事务的办法获得恢复能力的。这种系统将保障一个事务完全成功或者对数据库没有丝毫影响。这就要保证各事务的行为都是原子工作单元。

每个访问数据库的程序都可能是产生不一致性的根源。数据库系统通过规定一个所有程序修改都必须遵守的约束集合来防止这种错误的发生。如"雇员不能比他们的经理拿的钱多。"就是一个约束的实例。系统将会封锁所有违反约束的程序。约束通常体现为基于谓词演算的语言或规则集合。人们越来越感兴趣的是，把面向对象数据库的类型系统与这类约束知识结合起来。

最后一个需说明的面向对象数据库系统的特征是相联访问或查询。查询由定义在集合类型上的一系列操作组成。这些操作返回新的结构也都是原有数据库定义过的。关系型数据库在这点上已获得了圆满的成功。目前多数研究的焦点集中在面向对象数据库在这方面能否象关系数据库一样成功。

问题是面向对象数据库能否把查询优化处理扩充到其存储细节对外界是封装的或隐藏的那些实体上。因为查询可包括对用户定义的操作的任意组合，对优化程序说来要找到等价的转换是困难的。当实现被隐藏起来时，优化程序必须能够指出什么时候某个转换比原来不进行优化花费少。

与关系模型的关系

在商业数据库领域，关系模型代表了当代技术水平。因此，研究面向对象数据库与关系数据库的差别是值得的。

关系数据库以持久的数据空间观点出现。由整数、实数、串的初等量及以这些初等量表示的元组或元组集合（即关系）的结构化值所组成，（元组是个一维表，元组的集合构成一个二维表）。这种高级数据观点，对于主要产生报表的应用是很方便的。但对于象CAD系统或程序开发环境那样复杂的程序它却是个障碍。这类程序要求紧紧控制存储的利用。它们通常需要使用如栈、队列或字节流这样的数据结构。面向对象的数据库允许人们为那些复杂的任务所需要的数据结构创建对应的抽象。

关系型数据模型是基于值的，与前者不同，如CODASYL就是基于标识的数据模型。

它们的区别在于数据模型为相关对象相互关系提供了不同的机制。这正是任何数据库模型的基础部分。基于值的系统靠两个或多个相关对象上有相同（或类似）值来表达两个对象之间的关系。基于标识的模型可以不依赖于嵌入的值或它们所在的上下文而使两个或多个对象相关。

面向对象数据库系统享有象CODASYL这样的网状模型所具有的基于标识的特征。但仅从这种类似性无法断定两个模型本质上是一样的。面向对象模型增加了基于行为建模（即方法或操作）的抽象数据类型和以继承性的形式出现的增量修改机制，从而产生了与以前的网状模型颇不相同的模型。

尽管面向对象系统可做到基于标识的引用，但这不是模型内数据关系的唯一基准。Brown大学的ENCORE模型是通过性质使对象相关的。性质反映了对象的抽象状态。因此，性质P使对象X与对象集合S相关，且未陈述这个关系如何计算。它可通过直接引用S（或它的成员）的标识或通过连接操作匹配其它性质的值来进行计算。考虑下述类型定义：

```
Define Type Employee
  properties
    name: String
    dept: Department
    projects: Set of Project
```

dept是表达一个给定雇员所在部门的性质，可由一个直接引用类型Department的嵌入式对象标识符实现。换言之，如果Employee和Department两者的类型表示都是关系中的元组，则dept性质可由下述形式的关系查询实现：

```
dept(e) =Project((Join(Select
      (Employee, name=name(e)),
    Department), Department)
```

在这种方法中，面向对象数据库为基于值的和基于标识的访问提供了一个统一的框架。

面向对象数据库必须能够定义新的抽象并控制这些抽象的实现。从上述例子可以看到，在逻辑层保留相同的抽象时，有可能在实现层上将基于值和基于标识的关系合并。在这两种关系之间所作的具体选择会影响到某些查询性能。

实现的考虑

由于数据库所支持的应用性质不同，面向对象数据库的实现也有一些独特的问题。对于许多设计方面的应用，能否高效地遍历图的结构是非常重要的。如在电子CAD应用中的设计规则检查器，这种工具就要求，每当给出一个部件，系统必须能够很快地查出与它连接的其它部件。如果一个程序是计算电路板的，通常只要求计算该板连接元件的背面。虽然可把这类访问看成是一个退化了的查询，但对于这类查询，其它的实现技术可能比为整个大集合上查询设计的技术更有用。

在这种情况下，改善性能的一种方法是使遍历图中一个边所引起的磁盘故障的可能性最小。最好的办法是智能预处理。常用的方案是允许应用程序创建顺序地存储在磁盘上的任意大小的对象集合（也称为簇）。每当访问簇中的某个对象时，整个簇都被读（预处理）到该应用的存储中。

如果簇建立得正确，将会降低磁盘故障的次数。到底如何为一个特定的访问形式自动配置一个簇的集合，还需要进一步研究。

当面向对象数据库用于工作站网络时，必须指定一个或多个机器作为对象服务器，以便为应用程序提供所需的对象。在这样的系统中，必须使工作站和服务器之间的通信最小。智能高速缓冲存储对象多少与分簇技术有关，通常是一种减少通信的有效方法，其要点是让对象尽可能地接近他们的使用点。

编程语言中的对象指针由虚存地址实现。对于持久性对象，必须使用不依赖于对象的物理位置的指针，因为无法保证这个位置在不同的进程使用期间永远保持不变。一

般说来，应该使用系统产生的某些代用品。当不引用（dereferencing）这些指针时，通常要求从哈希表查找值去找到对象在磁盘上的位置。一旦找到对象并将它读到虚存后，必须消去查表固有的开销。已提出了种种方案：用虚存地址暂时代替基于磁盘的指针，然后当对象返回到磁盘上时，将指针换回为磁盘的代用品。

另一改进数据库性能的经典方法是引入辅助访问方法，可限制所需的查找量。索引通常用于有效地增强系统处理查询的能力。但对于面向对象数据库使用于方法的索引将会产生一些问题，除非能限制构造索引的种类，否则很难知道什么时候需要修改索引。

对于处理非常大的对象（如位映象）和管理大量的对象集合，索引结构是很有用的。Wisconsin大学设计的EXODUS存储系统为大对象构造了一棵树，使人们能非常有效地检索小片段。树结构允许人们不用读取整个对象就能从大对象中间访问一个字节序列。

查询处理

查询是对欲访问对象集合的高级规格说明。通常用一个特定的语言来说明查询。这个语言允许人们描述想要检索什么而不要求陈述如何做。查询处理器的工作是制定最有效的检索计划。查询语言（即第四代语言）以及与其相结合的优化器是关系方法中最主要的成就之一。

有人建议在现有的关系查询语言，如SQL中嵌入面向对象设施。也有人提出一组可用于聚集对象的操作，并为此提供一个面向对象的查询代数。

非第一范式关系（即在关系矩阵的单元中包含不止一个值）解决了以对象集合作为成分来表达复杂对象的问题。允许属性值是记录、向量或是另一个关系从而扩充了传统的关系模型。面向对象数据库解决了这个问题，但也引入了标识、数据对象和继承等概念。

面向对象模型中的查询可以算作是对引用某特定方法的一种表达式，而这些方法定义在聚集对象的类型（如Set、List、Tree）之上。作为一个简单的例子，类型Set定义了如下方法：

Select (S1: Set(T), P:
Predicate)→S2: Set(T)

其中Predicate是形如P (t:T)→Boolean的函数。输出参量S2是第一个输入参量S1的所有元素中能满足谓词P的子集，即：

Select(S, P)={S |S in S and P(S)}

Select操作是查询语言的一部分，类似于关系代数中的Select操作。

为什么面向对象模型中的查询与关系模型中的查询不一样呢？因为人们可以构造一个代数来表达整个对象集合上的查询。这些表达式涉及到一个优化问题：在关系模型中，查询是在非常简单结构上借助良定义的算子产生的（即规范化的关系）。但在面向对象模型中，查询会涉及到新定义的抽象数据类型的算子。每一个新类型通过引入新的操作就建立了一个新代数，而查询优化器却不知道它的性质。如果不知道这些新算子的代数性质，就很难将这个查询表达式转换成等价形式。

这个问题已在EXODUS项目中解决了。该项目在提供了转换规则以覆盖新的数据模型特征之后，对优化器作了扩充。这些规则将具有内部结点操作和叶结点存储数据的查询树转换为等价的查询计划。查询计划也是一棵树。它把低级访问例程作为内部结点，把顺序数据集合作为叶结点。

以抽象类型来封装实现对面向对象数据库的查询优化带来了新的问题。即使能产生查询的转换版本，也必须能判定处理这些查询表达式的相应开销。一般说来，处理开销取决于对象及其聚集在低层的存储结构，但很难根据方法的实现来确定。

例如，给定一个集合S是在某个属性A的B树上实现的，那么在属性A上检索整个S

可能相对地便宜。但知道了这种存储结构的存在似乎一开始就是对封装的破坏。

封装是一条构成良好软件结构的原则，它对保持应用层的模块是极为重要的。查询优化器应是数据库系统中可信赖成分，它可看到抽象数据类型的内部并决定实现方法。但若根据其它的类型做出实现，如何能有效地管理它们也仍然是个问题。Graefe和Maier提出了一个叫做展现的技术，即抽象数据类型可对优化器展现其实现的细节。所展现的行为以表达式给出，而表达式与按低层实现类型作出的查询片段是等价的。

事务

为保持数据库在并发执行进程中的正确性，数据库系统定义了原子事务概念。当允许并发执行时，这些事务是工作单元，以便保证产生的结果与顺序执行的结果相同。任何保持这个性质的交替操作则看作是可串行的。

已提出不少可保证串行执行的实现。多数基于读／写语义。也就是把对数据项X的读和写均定义成与其它对X的写相冲突。数据管理者就可以在保持可串行性的情况下决定读、写进程的调度。

面向对象数据库提供了改进传统方法的机会。按面向对象方法，数据库对正在执行的操作要知道得更多些，不只是读或写，而是具有更多的语义。例如，对于一个队列数据类型，可能有象enque(插入队列)和deque（删除队列）这样的算子。从某种观点上来说，它们可分别地看成是读和写，但如果考虑到这些算子的特定意义，就会获得更高的并发性。

假定有一个队列对象Q和两个事务T1和T2。如果T1在Q上实行了enque，则T2不能用公共读／写语义做deque直到委托了T_1为止。但是请注意，对于非空队列，这两个算子彼此之间不影响对方的结果，它们可以无冲突地做下去。

对于设计环境中所见到的这样类似的协作应用，可串行性概念是一个非常强的正确性准则。设计者通过使用面向图形的编辑器来和该环境中许多对象打交道。如果把一个编辑器（或一组编辑器）的处理期看作是一个事务，就可用这个办法得到可串行性。设计者并未使之串行，相反他们彼此之间都在未完成状态下共享信息。

此外，一个单一事务T可涉及许多对象，而这些对象又通过复杂的综合约束与许多其它对象联接。如果T要检查并调整这些对象的状态使它们保持相容，则T必须对它们全部加锁（读与／或写）。这就降低了大型设计事务之间可能并行的数量。研究人员正在开始考虑提出允许用户改变系统给定的正确性准则的模式。

分布式对象

当数据库用来支持设计工作站网络时，必然会遇到分布式数据库环境所引起的一些问题。为简化程序设计并保证数据独立，分布式数据库要力争数据分布是透明的。用户应能命名数据，其方法与集中式数据库中用到的相同。系统负责定位所需要的数据项并自动地进行更新。用户只须关心逻辑问题。当数据在整个网络中重新分布时，程序应保持不变——系统根据当前的数据位置，为处理要求不同节点数据的查询产生一个新的优化。

在面向对象数据库中，程序（准确地说是方法或操作）经常被看成是对象，因此可象其它对象一样在分布式数据库中移动。当要实施一个计算或处理一个查询时，系统可以选定，是将数据移到程序中还是将程序移到数据中。若在一个非常大的对象X上执行一个方法M时，通常将方法移到X驻留的机器上更为合理。

在分布式系统中也可使用高速缓存技术。当对象从一个机器移到另一个机器上时，在一段时间内保留当地的备份通常可缩短以后的检索。

如果一个对象放置不当，面向对象的分

布式系统的解释性质会产生尖锐的性能问题。将方法名晚期汇集到方法体上时，需查看好几个对象才能解决哪些代码需要进行。仔细地安置对象和使用智能计划程序能使通信最小，这对面向对象的分布式数据库的性能有极大差异。计划程序决定了操作顺序、执行这些操作的机器、以及接收结果存储单元。

在分布式环境中由不同种类的系统组成的网络是一个问题，虽然这些系统必须一同工作，但对各参与系统的特征控制不多。这种异质性表现为几种形式：参与工具和系统的底层数据格式不同；用于开发应用的语言不同；及设计者需要共享的信息不同。

面向对象系统中的抽象机制可用于为已有数据包建立桥梁，使之成为实现新抽象类型的工具。这种新类型往往用外部数据包支持的数据结构表示。每当引用这种新类型的方法时，其方法代码将调用这个数据包以访问外部的表示。一个旧的应用总可用它持有的同样方法去访问和更新持久性数据，而新的应用对它的访问必须通过面向对象模式（图1）中定义的抽象类型。尽管这种方法可能慢一些，但能访问散布在不同存储系统上的数据，这种能力通常是必需的。

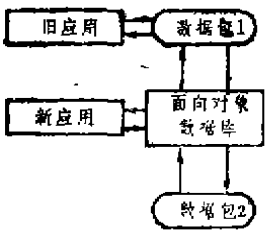

图1

面向对象数据库用抽象来保证旧应用与新应用之间的相互可操作性。旧的应用可照旧访问它们的数据包。新的应用利用面向对象数据库提供的抽象机制既能访问旧的也能访问新的数据包。

其它特征

有关面向对象数据库的文献经常讨论那些在真正的面向对象系统中不需要而在 CAD 应用的数据库系统中有用的特征。这包括版本控制、复杂对象和大型（与协同）事务。这些文献更多地讨论了设计环境的应用能力，而不是面向对象的固有属性。

面向对象数据库中的版本管理设施把对象看成是时间序列上的一组快照。然而在实现版本管理时要涉及到若干问题。一个是版本集合的基本结构。数据库必须能够处理二个以上进程同时想要更新同一对象的情况。一种解决办法是将版本分支，特别是当相互竞争的那些版本在一些基本的假设上有冲突时。另外，可能还要提供合并分支版本的机制。

其它一些问题是如何引用对象版本集合的成员。一种方法是用版本号静态地引用它们。另一种方法是动态机制。利用函数返回一个特定的版本号。这个函数可在不同的时间返回不同的版本号。

复杂对象是由其它对象建立的一种对象模型，复杂对象设施的难点是 part-of 关系的语义，即成员对象与复杂对象间的关系。这个领域的研究涉及到如何使数据库系统的附加行为归结到会影响其它操作行为的part-of 关系上。例如，当删除一个对象时，也希望删除它包含（即与part-of 关系相关的）的所有对象。关于复杂对象优化访问的细节请参阅有关文献。

非第一范式关系提出了以结构化对象成分表达复杂对象的问题。通过允许一个属性值是一个记录、向量或其它关系，非第一范式关系扩充了传统的关系模型。

小结

面向对象数据库是为满足复杂设计应用中数据处理的需要而设计的。当这些系统的数据建模设施与面向对象程序设计语言相象时，数据库系统就可将持久性、并发控制、可恢复性、一致性管理和查询语言嵌入其中。

此外，面向对象数据库可设计一些略不同于其对应语言的数据模型，以便有效地支持数据库特征。这方面的一个例子是将聚集（即集合）合并到系统的模型方法。这些集合是形成有效查询的基础。

数据库管理的历史也就是数据模型相互竞争的历史。每种模型都各有千秋，面向对象的方法提供了一个基于抽象的模型，并允许类型设计者使用在基本功能实现上最能适合于它们的应用的技术，从而一统群芳。尽管许多研究问题尚待回答，但面向对象数据模型抓住了当今日益增长的复杂环境提出的先进数据处理这个问题。

［李卢凡、何玉洁、林守江译自《BYTE》1989年9月］