

## 第3章 为什么使用对象建模

在关系数据库设计中，用来创建数据库逻辑模型的标准方法是使用实体关系模型（ER模型）。可以利用ER模型的灵活性来创建健壮的数据模型，但是关系模型在体现与数据相关的业务规则上提供的词汇相对较少。

随着面向对象技术的发展，出现了其他一些建模技术和符号。并不是因为对象建模是当前的一个趋势，为了追求时髦才去拥护它，而是因为许多建模人员包括作者都感觉到了使用ER建模结构所带来的障碍和局限性。ERD不能提供足够丰富的专业术语来适应新的结构。现在是重新调整数据建模方法的时候了，有三种选择：

- 扩展ERD的符号来支持面向对象。

- 逻辑设计时使用现有的ERD符号，而建立物理数据模型时使用更多的选择。

- 使用UML或其他建模符号。

第一种选择是可行的，但是需要大量的工作来支持面向对象的扩展。有人会认为 UML可作为ERD符号的扩展，但这需要在ER建模的思维方式上做很大的变化。

第二种选择也没有任何错误。如果ERD已经足够好了，Oracle和其他提供商就没有理由不采用这种方式。但这并不是问题的全部，因为即使你没有计划使用 Oracle 8的对象扩展，但UML图表提供的一些功能确实比ERD有更大的优势。UML同ERD一样是一种逻辑建模工具，没有理由在表及Oracle 7数据库的传统方式中不使用UML图表。事实上，各种原因都会促使你使用它。虽然你的工作可能因为政策或技术的原因无法升级为 Oracle 8或不使用 Oracle对象，你仍然可以使用UML来设计数据库。

Oracle 8是 Oracle 7的升级版。Oracle 8不仅提供了关系对象特性，也提供了许多新的特性，即使不使用关系对象特性也应当移植到 Oracle 8环境中。

Oracle 8的对象扩展是将ERD转变为UML的明显推动力。利用当前 Oracle 8存在的对象扩展，以及与 Oracle 8后面的版本保持一致的面向对象结构的完全应用，UML（或其他建模标记）都将需要使用。第三种选择即使用UML或别的等效的建模标记应是最好的方法。面向对象理论使建模方式发生了重大变化，应当利用这些理论的进步。

UML作为一种建模语言的标准，很大地促进了UML的使用。关系和对象方法都存在很多建模工具，面向对象的研究人员带领开发着一种通用建模语言，以使每一个人都可以使用这种语言。即使数据库的研究人员已完全掌握了UML，在看法上也会存在一些区别。因此，采用通用行业标准的优势是不容忽视的。

### 3.1 实体关系建模的局限性

了解现存标准并没有固有的错误是非常重要的。ERD已为工业的发展尽心服务超过10年了，许多建模人员选择使用ERD是因为它的简便实用和雅致的风格，如果数据建模人员对本书提出的这种总体结构和使用Oracle 8对象扩展不感兴趣，那么传统的ERD对他们的需要来说可能已经足够了。

但是,从更广的角度来看,ERD建模方式存在着一些问题。ERD存在的问题首先体现在它的命名上。ERD建模的中心思想是,可以仅通过实体和它们之间的关系合理地体现一个组织的数据模型。但从表面上看,这样做似乎对描述一个组织的信息过于简单化,并且词汇量也远远不足。UML的一个重要优势在于增加了处理信息。即使不从这一点考虑,ERD建模在关系数据方面仍然存在着其他严重局限性。

“联合”或“交叉”实体明显是两种不同类型的实体,它们出现在多对多关系中,但是用传统的ERD无法表示这些区别,同样也无法区分时间相关实体(即时间是唯一标识符的一部分)。虽然在多种情况下,这种区分可以增强图表的功效。

本节指出了ERD建模的许多局限性,希望读者能理解为什么迫切需要使用更加灵活、健壮、面向对象的UML来代替当前ERD建模这种工业标准。

## 选择UML

无论如何,UML是对象程序员而不是数据建模人员的成果。互联网上的评论表明许多对UML感兴趣的人对数据建模缺乏经验,大部分关注UML的思想和注意力都集中于用C++或SmallTalk语言来支持结构。这样,并没有一个完善的数据建模工具,目前拥有的可用于数据建模的工具是由那些更关注于面向对象编程而非数据建模的人员设计的。

互联网上列出的观点和其他的一些设计人员认为UML根本就不是一种数据建模工具,他们认定UML不适于数据库设计。显然事实并非如此。UML可以完成ERD所能做的所有建模工作,而且UML可以描述ERD所不能表示的关系。可以得出这样的结论:UML并不是十全十美的,但它作为建模工具比ERD有更大的灵活性。不能肯定UML可以满足建模的所有需求,我们仍会面对一些无法用UML数据模型很好地表示与数据相关的业务规则的情况,但UML确实提供了一个更为灵活的系统,使我们能够精确地表示大多数与数据相关的业务规则。

## 3.2 使用UML

对于熟悉ERD建模的用户来说,转向UML并不是十分困难的。许多建模人员都非常熟悉实体关系实例的限制,使用UML后一些限制就不存在了。但是ERD的一些不足之处仍存在于UML中。

紧记UML中“类”(对象组)一词代替了ERD建模中使用的“实体”。“对象”在文字上的意思与这里表示的含义有一定的区别。下面是不同描述的“对象”的定义:

“你可以像对待一个事物那样来对待它。对象具有状态、行为和标识符;对象相似的结构和行为定义在它们的公共类中。实例和对象这两个词是可以互换的。”出自《Object-Oriented Analysis and Design With Applications》,作者Grady Booch, Addison-Wesley出版社,1994,第516页。

“面向对象系统用于信息存放的单元称之为对象,对象包括数据和一组用来控制数据的操作。对象可以是一张发票、一个装满东西的橱柜、一种雇员类型或是一台喷气发动机的一部分。每一个对象都包含使它做相应处理的数据和逻辑操作。例如,发动机对象包含发动机的各种特性,而软件决定了这台发动机所能做的工作,比如向某个方向旋转。”出自《Paradigm Shift》,作者Don Tapscott和Art Caston, Osborne/McGraw-Hill出版社,1993,第172页。

“对象是计算机系统中的一个元素，有一个唯一的标识符，状态（由公共和私有的数据体现）以及共有和私有的操作（方法）体现对象的行为。”出自《Object-Oriented Software Engineering》，作者George Wilkie, Addison-Wesley 出版社，1993，第386页。

“对象是一种独特的可以确认的事物，系统必须在对象中存取数据以便执行系统的基本功能。”出自《Analyzing Systems》，作者James A. Koval, Prentice-Hall出版社，1988，第184页。

“对象是活动组件，通过一定的信息或事务触发，产生相应的动作。”出自《Foundations of Business System》第2版，作者安达信咨询公司, Dryden 出版社，1989，第233页。

“对象是逻辑数据库描述的组件，表现真实世界的实体需要被存储的信息。”出自《Information Engineering: Book —— Planning & Analysis》，作者James Martin, Prentice-Hall出版社，1990，第474页。

“对象是面向对象程序中的一种结构，包括一个封装的数据结构和数据方法。这些对象被排列成层次结构，以使得对象可以从父类中继承方法。DB2中对象指的是数据库、表、视图、索引和其他结构。”出自《Database Processing》，作者David M. Kroenke, Prentice-Hall出版社，1995，第583页。

“对象在面向对象方法中是指具备相关信息或行为的个人、地点或事物。”出自《Designing Quality Databases with IDEF1X Information Models》，作者Thomas A. Bruce, Dorset House出版社，1992，第535页。

注意到除了方法——它可以以PL/SQL、Java或C/C++的函数和过程来实现，对类（或对象组）的定义同实体的定义是相同的。特别是，上述第6项同我们对实体的定义完全相同。类的正确定义要比实体的定义涉及的范围更广。也可以使用对象来描述处理信息，但是，这样“对象”将被用在数据对象的内容中。类是一个组织中所关心的部分或是划分感兴趣的事物的一种方法，其核心是实体或类经常用来体现真实世界中的某些事物。

我们对类的定义比面向对象领域中对类的定义范围更小一些，出现这种区别的原因是，面向对象领域是从编程环境（经常是C++或SmallTalk）的角度得出这个概念，而数据库领域中考虑的是建库。建立的逻辑模型的结构是如何实现的无关紧要，而且事实上，它的实现并不影响我们的思考方式。从C++程序员的角度来看，类是定义的数据结构，对象是这些定义的数据结构在运行时的实例。而在数据库领域中，类与表相对应。

既然得出的概念来自不同的角度且解决不同的问题，定义的内容自然就会不同。在Bertrand Myer的《Object-Oriented Software Construction》（Prentice-Hall出版社，1997）一书中，定义对象时首先定义了一个抽象数据类型（ADT）：一组特定的数学元素，对所有这些元素列出适当的函数和这些函数的正式属性（术语表第1193页）。这样，类就是“部分或全部实现的抽象数据类型”（术语表第1194页）。虽然这是从文章中提取出来的两句话，但很清楚地能看出这个定义是来自面向对象编程的角度而不是数据库设计的角度。如果是设计对象关系数据库，这里的定义可能更加合适。

### 3.3 ERD建模及其局限性

使用ERD建模存在三种可能的关系类型：

基数关系。

依赖关系（UID连线）。

子类/超类。

上述关系类型对用来描述复杂数据模型的无数个可能的关系没有提供充分的灵活性。用UML表示这些关系时将分别讨论每一种关系的局限性。

### 3.3.1 基数关系

两实体之间所有基数关系模型可以描述为：

“实体1的实例N1与N2之间的某些实例同实体2的实例N1与N2之间的某些实例存在关系。”

表示模型中全部基数关系的概念并不是UML中的新概念，即使在一些早期的ERD技术中也使用了相似的表示方法。巴克使用“crow's feet”一词来表示这种基数关系，这样似乎更好理解一些，但还是存在了很大的局限性。用巴克的表示方法描述基数关系就是：一个实体的第0、1或N个实例与另一个实体的第0、1或N个实例存在关联。在Oracle的ERD中，基数关系的表示极为有限，这使得无法表示许多与数据相关的业务规则，下面将比较ERD与UML处理基数关系的区别。

使用ERD和UML表示一对多关系非常容易，如图3-1第1行所示，UML中表述为类A的0或1个对象与类B的任意数量的对象相关联（\*号是0..N的简写）。但是，在实际应用中除了\*，0..1、1、1..\*的基数关系就很少使用了。图3-1第2行表示的基数关系很少使用。除了以上两种基数关系，一般用不到其他基数关系。再举出更多业务规则的例子，可能会碰到其他基数关系。在分析的早期阶段也许倾向于更多地使用其他基数关系，但是当模型成熟并进入应用设

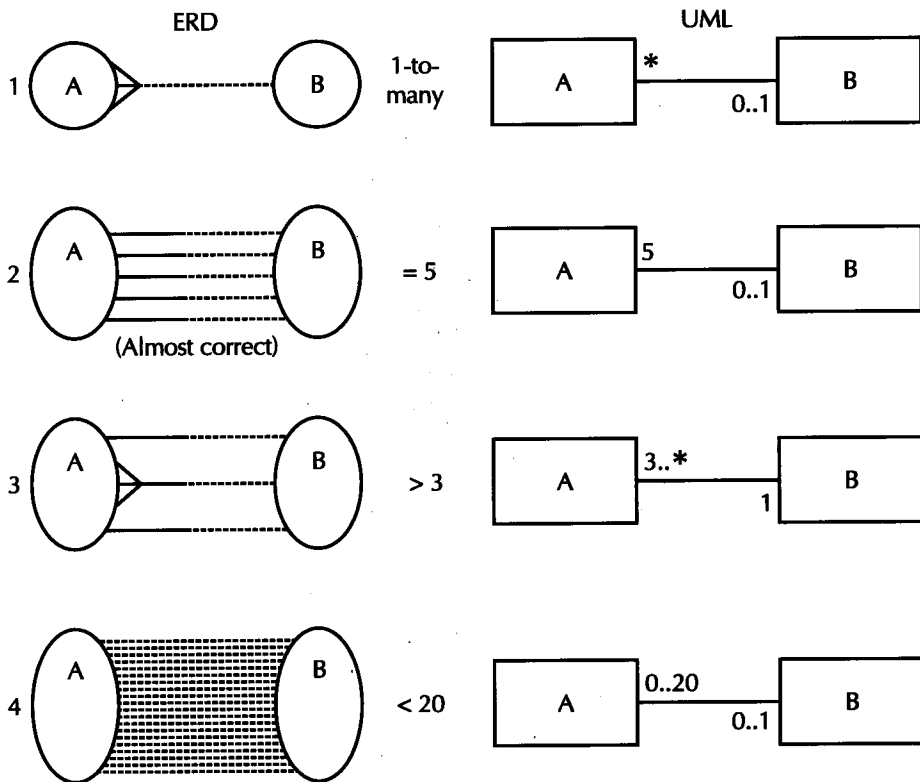


图3-1 基数关系

计时，将会发现在数据库中不应该实现这些约束，因为现实条件下违反这些规则的情况确实存在。例如，如果一个篮球队至少必须有 5 名球员，在创建数据库时，如何将第一个人增加到球队数据库中呢？除非能够在数据库中定义数量明确的插入状态，且这些插入状态同属于一个事务。许多这种过于复杂的限制关系无法轻易实现，虽然 UML 中存在更加灵活的基数关系，但在实践中很少使用。

其他基数关系无法方便地用 ERD 表示，试图表现这些关系将导致回旋表的产生。例如，如果业务规则是篮球队中必须有 5 名球员，使用 ERD 表示这种业务规则是相当困难的，而 UML 可以简单地完成这种表示，如图 3-1 第 2 行所示。

对于业务规则“一个委员会必须至少包括 3 名成员”，利用 ERD 描述这个规则显得较为笨拙且无法很好描述。用 UML 表示就比较方便，如图 3-1 第 3 行所示。

对于规则“一个委员会不能超过 20 个人”，使用 ERD 就不可能清楚地表达，用 UML 表示非常简单，如图 3-1 第 4 行所示。

### 3.3.2 子类

ERD 中的子类是指将一个实体分成若干个相互独立而整体内容全面的子集。举一个最普通的例子，雇员的工资发放既可以小时发放也可以月发放。

乍一看来，当分成子类时 UML 语法与 ERD 语法相同，如果只看职员工资发放方式的例子，并看不出 UML 语法比 ERD 语法明显优越的地方，如图 3-2 所示。

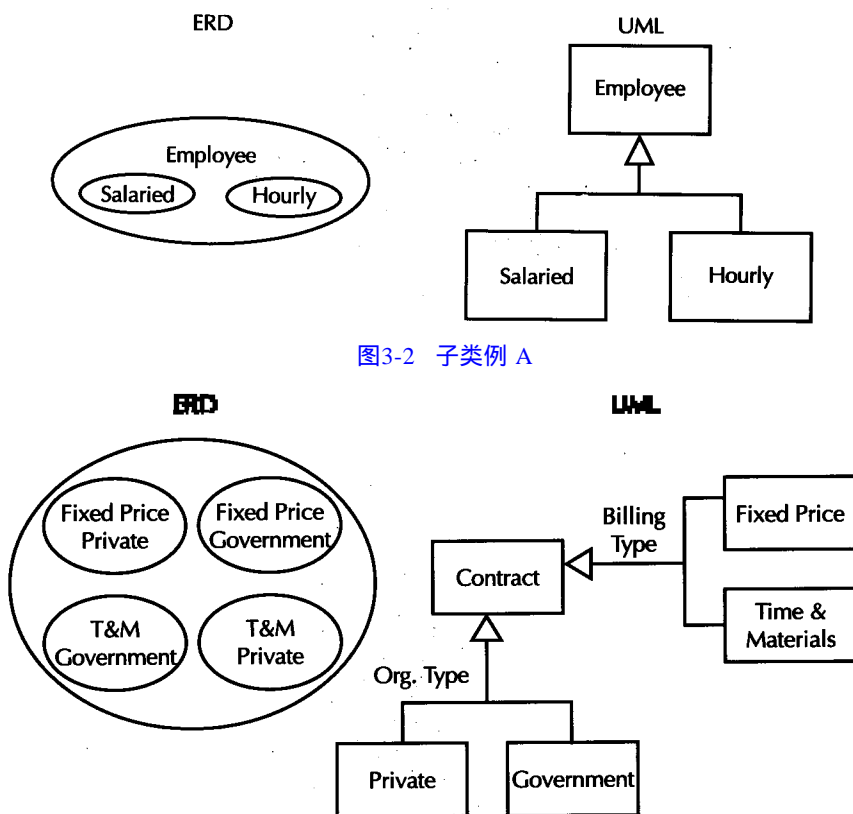


图3-2 子类例 A

图3-3 子类例 B

但是，使用UML语法，可以同时用多种方法表示一个被分成子类的特定类。例如一个关于合同（Contract）的类，可以用固定价格（Fixed Price）、时间与材质（T&M）、属于政府的（Government）还是私有的（Private）来划分。用UML表示很方便，如图3-3所示。

用ERD表示，需要所有可能的子类的笛卡儿积，在上例中是可能的。但如果每一个子类有五个或更多的值，笛卡儿积将变得很大。

### 3.3.3 多对多关系

在表示多对多关系时，ERD的能力也有限。在ERD中，通过两个一对多依赖关系的相交来表示多对多关系。而使用UML，可以用更为自然的符号来标识明显属于这个关系的交叉实体。如图3-4的UML 1图所示。

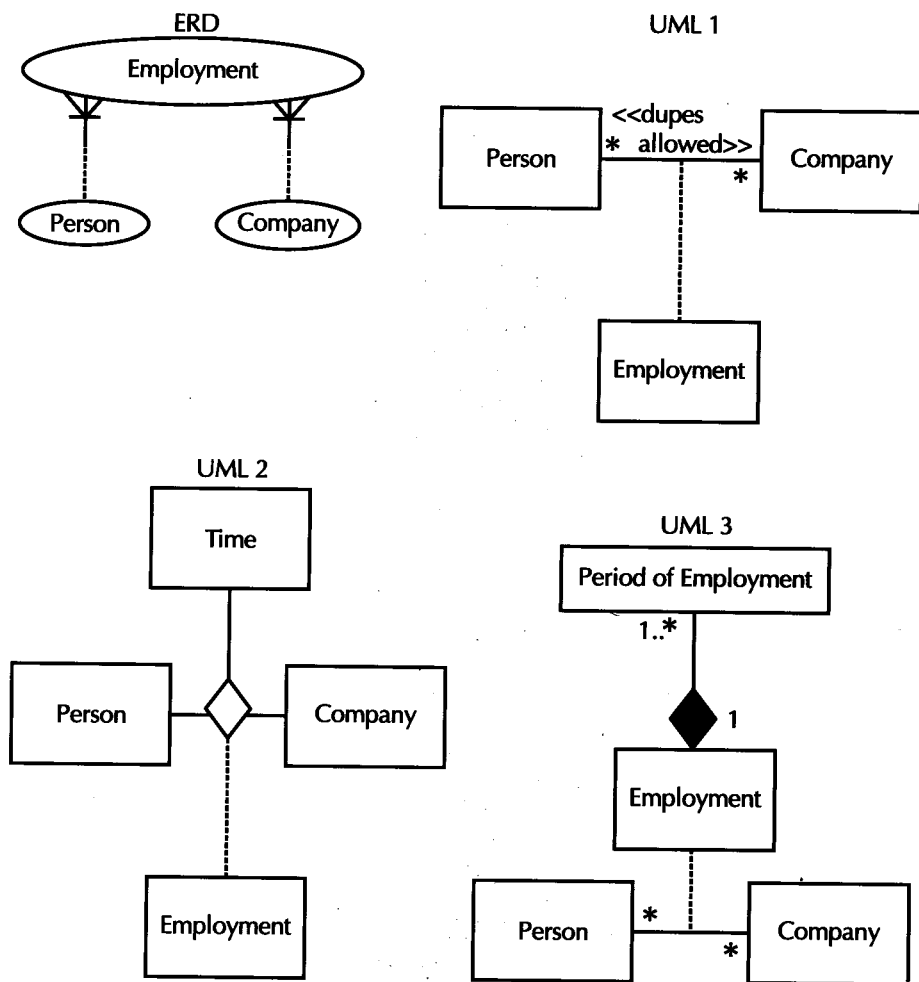


图3-4 多对多关系

UML中，交叉实体被称为相关类。根据UML理论，这种相关类是一种正式交叉，它的唯一标识符正是每一个附属类中的一个对象。这样，它提出了一个不切实际的约束，即一个雇员能且只能与某一个公司建立联系。Oracle的对象数据库设计器中还不存在这种相关类结



构，希望在实现时，Oracle不会坚持这种约束所需的标准。

所幸的是，UML提供了一种扩展建模符号的方法。注意在图 3-4的UML 1中表示多对多关系的连线上方，增加了“<<dupes allowed>>”，这个符号被称为模板。模板允许对 UML结构重新定义或修改，UML支持扩展的能力是它最大的优点之一。可以用 UML在形式上把这个概念表示为图 3-4中的UML 2 或UML 3，但这两个并不是最好的模型。用这些模板产生正确的代码需要像Oracle的ODD这种工具的明确支持。否则，模板只能提供文档信息。

### 3.3.4 多重分类

可能存在一种特殊的类，它由两个或多个其他类组成。例如，一个人可以既是一个消费者又是一名雇员。在 ERD中，这种现象称为非唯一子类。建立这种标准的面向对象的分类方式，用来处理由各种数据类型组成的所有属性的实体；或是处理分开的实体，每个实体结构使用可选的强制性一对一关系。如图 3-5所示。

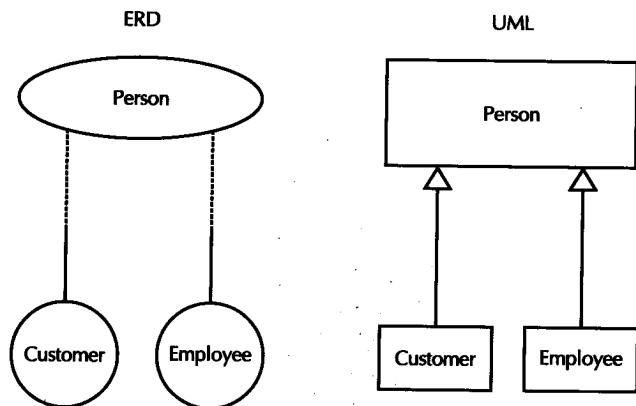


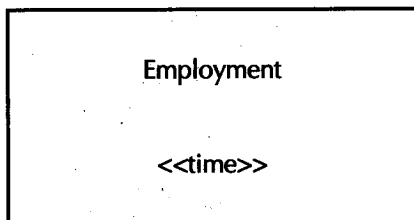
图3-5 多重分类

表示这种关系的UML符号就稍微清楚些，并且清楚地显示了类之间的分类关系。

## 3.4 使用UML图式的缺点

如前所述，UML并不是解决所有问题的万能钥匙。它没有解决 ERD建模出现的所有问题，但确实使完成任务更加容易些。UML也存在一些缺点，如不使用固定形式仍然无法表示时间相关实体，这种时间相关实体是重要的一类类型，正像它们是一类重要的实体那样，需要特殊的处理。需要能够直观地区别出与时间相关的实体，这个在 ERD中出现的问题UML仍然没有解决。

UML确实提供了扩展功能，这种扩展在 UML中是允许并支持的。表示时间相关实体的正确语法如下所示：



这个图表使用模板“time”作为雇佣(Employment)类唯一标识符的一部分。

### 3.5 面向对象方法的优点

前言的最后曾给出了一些例子，展示了利用面向对象的思想怎样改善模型。需要指出的是，在这些例子中，还没有建立 Oracle 8的结构或使用UML，而只是利用了面向对象的思想。这些例子暗示了这样的事实，即没有必要等待成熟的面向对象数据库的出现，可以直接利用面向对象思想和技术。在这些设计中，UML可以通过给出更好的专业术语来提供帮助，但必须在一开始就建立面向对象的结构。

使用UML设计面向对象数据库并不比使用 ERD开发第三范式数据结构提供更多帮助。使用UML建立结构不好的数据库与使用 ERD一样容易。此外，因为UML产品例如Oracle的对象数据库设计器（Oracle设计器的UML部分）将自动产生一些复杂的对象结构，这样建立的数据库比相同情况下利用 Oracle 7.3建立的有局限性的结构似乎更加无用。

### 3.6 Oracle的Object Database Designer

Oracle的Object Database Designer（ODD，对象数据库设计器）产品使用UML的部分实现来帮助设计对象关系数据库。该设计器作为Oracle设计器的一部分来帮助设计者构造 Oracle 8 DDL。

目前，这个产品只是部分实现了UML。设计它主要作为帮助产生 Oracle 8对象结构的工具，而不是作为一个完全的UML建模工具。没有包括一定数量的关键结构，但包括多重分类、交叉类和用于查表的动态分类。尽管如此，该产品是一个很好的尝试，并能用于设计对象关系数据库。

### 3.7 结论

在建模表示方面虽然存在一些缺陷，UML还是向前迈出了有意义的一步。不仅可以用UML表示更多的数据相关的业务规则，该表示法也鼓励我们在设计中注意思考如何利用更多的面向对象方式。虽然Oracle的对象数据库设计器没有完全实现UML，但它也比使用ERD建模前进了一大步，应该得到使用。

必须自己决定是否UML图表比ERD更为先进。对模型来说，这更依赖于总体结构，我们相信UML比ERD具有更大的优越性。