第13章 语义建模

13.1 引言

自从20世纪70年代末期以来，语义建模就成为一个研究的课题。这项研究的动力——即研究者试图解决的问题是，典型的数据库系统对数据在数据库中的含义的理解是非常有限的：它们通常可以“理解”某些简单的数据值，以及这些值的某些简单约束。但是对其它方面的理解却很少（所有复杂的解释都要用户自己去完成）。显然，系统理解得越多越好[⊖]，这样它们就会对用户的交互行为反应更智能化，并且会支持更复杂的用户界面。例如：SQL应该能明白零件的重量和发货的数量在含义上的区别。虽然这两个都是数字值，但在类型上是有差别的——即在语义上是不同的。因此，在这种情况下，如果对零件和发货量按照重量和数量相匹配的条件进行连接，那么它即使没有被立即拒绝也至少应该受到质疑。

当然，这个具体的例子与域的概念非常有关系——这个例子显示了现在的数据模型并非完全与语义特征无关。例如，初始定义的域、候选码和外部码都是关系模型的语义特征。换句话说，这些年来发展的注重语义问题的“扩展”数据模型只是稍微地比以前的数据模型增加了一些语义而已（解释参见[13.6]中Codd的文章）。理解数据含义是一个永远不会停止的任务，这个领域也会随着人们对其理解的加深而不断地发展。语义模型这个概念经常被用作代表一个或多个“扩展的”模型，并非由于它易于代表这种模型，而是因为它有助于模型理解所涉及的问题的所有语义。另一方面，“语义建模”是对试图表示语义的所有行为的一个恰当称呼。在这一章中，我们首先对这些行为中潜在的一些思想进行简单的介绍；然后深入地探讨一种具体的方法：即实体/联系方法（实际中最常用的方法）。

语义建模有许多称呼，包括数据建模、实体/联系建模、实体建模和对象建模。之所以称之为语义建模有以下原因：

- 不用“数据建模”这个概念是因为（a）“数据建模”这个概念与前面所提到的用结构、完整性和操作等组成的正式系统的“数据建模”概念相抵触，（b）数据建模这个概念有可能会导致一种普遍的错误思想，即，一个数据模型只包括数据结构。注意：这与在第1章1.3节中所提到的数据模型概念有两个不同含义的说法是相关的。前面的概念是通常的数据模型（在这个意义上关系模型是一个数据模型）。而第二种则是一些具体的企业的固定的数据模型。在本书中并不包括第二种含义，而在其它书中是可能包括的。
- 不用“实体/联系建模”的原因是，它只是表示这种问题的一种具体的解决方法，而在实践中许多其它的方法也是可能的。但是“实体/联系建模”这个概念已经很常见并且应用得非常广泛。

[⊖] 不用说，在第8章讨论的支持关系变元和数据库谓词的系统是能“多理解一点语义的”。也就是说，这种谓词支持是语义建模正确而且恰当的基础。然而可悲的是绝大多数的语义建模方案都不基于任何这样的基础，而是特别到某种程度(参考资料[13.17~13.19]中的提议是一个例外)。然而，由于商业界对商业规则[8.18~8.19]越来越多的重视，这种不好情况可能会改变。第8章中的谓词在这种意义上讲只是基本的“商业规则”。

- “实体建模”这个概念并不是不好，但是它看起来比“语义建模”更具体一些，因此可能会有某种不适当的暗示意义。
- 对于“对象建模”这个说法的问题是：“对象”在这里是“实体”的一个同义词，然而在其它上下文中则可能会是另一种含义（其它的数据库上下文环境，这方面问题可以参见本书的第六部分）。实际上，看来这个现象（有两个不同的含义的现象）很可能引起在本书其它部分提到的第一大错误（The First Great Blunder）[3.3]。请参见第25章对这个问题的详细讨论。

下面回到本章的主题。把这一章放入本书的这一部分的原因如下：语义建模的思想在数据库设计中，即使在缺少直接的DBMS支持的情况下，也非常有用。因此，数据模型在商业上得到实现之前，最初的关系模型思想是作为最初的数据库设计目标的应用而产生的，因此即使没有商业上的实现而是仅作为设计的补充，这些“扩展的”数据模型思想也是十分有用的。事实上，在撰写本书的时候，语义建模的思想已经在数据库设计领域产生了重要影响，这样说可能并不为过——几个基于某些语义建模方法的设计方法已经被提出来。由于这个原因，本章的重点具体来说是语义模型思想对数据库设计的应用。

本章的计划如下。在这一节之后，13.2节解释语义模型中常见的术语。13.3节介绍最知名的扩展数据模型：Chen的实体/联系（E/R）模型，在第13.4节和13.5节介绍这个数据模型在数据设计中的应用（其它的模型作为对“参考文献”中某些文献的注释进行了简单介绍）。最后，13.6节提供了对E/R模型的某些方面的简单分析，13.7节是小结。

13.2 总体方法

下面根据四步来描述语义建模问题的总体方法：

1) 首先，要辨别一组语义概念，这些概念在非正式地讨论现实世界时看起来很有用。例如：
   - 世界是由实体(entity)组成的（虽然不能详细精确地刻画什么是实体，实体的概念至少在讨论现实世界时看起来确实是有用的）。
   - 进一步讲，实体可以划分为实体类型（entity type）。例如，所有的雇员都是雇员实体类型的实例(instance)。这种分类的好处是所有给定类型的实体都有共同的特性(property)——例如，所有的雇员都有薪水——因此这样可以明显地简化表述。例如，在关系中，共同点就可以被归到关系变量的标题（heading）中。
   - 更深入地讲，每一个实体都有一个用来识别自身的具体特性——即每一个实体都有一个标识（identity）。
   - 再深入一步，任何实体都可以通过联系(relationship)来与其它的实体相关。

   如此等等。但是注意：所有的这些术语（例如，实体、实体类型、特性、联系，等等）都不是精确而形式化定义的——它们是“现实世界”的概念，而不是形式化的概念。第一步不是形式化的步骤，然而，下面的2~4步却是形式化的。
2) 设计一组相应的符号化的对象来代表前面的语义概念（注意：对象这个术语并不代表任何已存在的含义）。例如，扩展的关系模型RM/T（参见[13.6]）提供了一些特殊的关系类型称为E关系和P关系。严格地说，E关系代表实体而P关系代表特性；如上面所讲的，实体和特性并不是形式化定义的，然而，E关系和P关系却理所当然是形式化定义的。

3) 设计一组正规的、常用的完整性规则（或者“元约束（metaconstraint）”，这是在第8章用到的术语）来搭配这些正规的对象。例如，RM/T中包括一个称为特性完整性的规则，这个规则要求进入P关系必须有进入E关系的许可（数据库的每一个特性都必须是某些实体的特性，这个要求就反映了这个完整性规则）。

4) 最后，设计一组用来操作这些正规对象的正规操作符。例如，RM/T提供了一种属性操作符，这个操作符可以用来将E关系及其所有相关的P关系连接在一起，而不必知道有多少个这样的关系和这些关系的名字是什么。因此，可以将任意实体的所有特性都收集在一起。

上面2~4步中的对象、规则和操作符一起组成了一个扩展的数据模型——“扩展模型”，这就是说，这些构造确实是一个基本的模型如关系模型的超集；但是，在此上下文中并没有对于什么是扩展的和什么是基本的之间的明显区别的解释。请注意，规则和操作符与对象一样，都是模型的一部分（就如它们在关系模型中一样）。另一方面，从数据库设计方面看，较为恰当地说，对象和规则比操作符更重要；因此，本章中其余部分的重点就是对象和规则而不是操作符，偶尔可能会就操作符的方面来进行一些讨论。

步骤1中包括辨别一组语义概念，这些概念谈论现实世界时是有用的。有几个这样的概念——实体、特性、关系和子类型——已经在图13-1中使用正规的定义和例子显示出来了。这个例子是有意选取的，它显示了在现实世界中同样的对象可能会被一些人当作实体，而被另一些人当作特性，还会有人将其当作联系（顺便说一下，这个观点说明了为什么不可能给实体这样的术语下一个准确的定义）。支持灵活的解释是语义建模的目标，一个不可能被完全达到的目标。

概　念	非正式定义	举　例
ENTITY(实体)	可相互区别的对象	供应商，零件，发货 雇员，部门 人 乐曲，协奏曲 乐队，指挥 购货订单，订货明细
PROPERTY(特性)	表示实体的一项信息	供应商号码 发货数量 雇员所在部门 人的身高 协奏曲类型 订购日期
RELATIONSHIP(联系)	充当两个或多个实体间联系的实体	发货(供应商-零件) 分配(雇员-部门) 录音(乐曲-乐队-指挥)
SUBTYPE(子类)	如果每一个*Y*必是一个*X*，则实体类型*Y*是实体类型*X*的子类	雇员是人的子类 协奏曲是乐曲的子类

图13-1 一些有用的语义概念

顺便提一下，注意在下列术语之间很可能会出现冲突：（a）语义层用到的术语，例如在图13-1中所列出的术语，和（b）在某些潜在形式——如关系模型——中用到的术语。例如，

许多语义建模方案中用到了属性一词来替代特性——但并不是说这个属性与关系层的属性是同样的概念，也不是意味着它是关系层属性的映射。举另一个例子来说（很重要！），用在E/R模型中的实体类型概念并不与在第5章中讲的类型概念相同。更具体地说，这样的实体类型在关系设计中很可能映射到关系变量中，所以它们当然并不与相关的属性类型（域）相一致。但是它们也并不与关系类型一致，因为：

1) 一些基本关系类型有可能在语义层与联系类型而不是实体类型相符合。

2) （大致上说）来自关系的类型可能并不与任何语义层的任何类型一致。

层次之间的混乱——特别是在名词冲突中出现的混乱——在过去已经导致了巨大的错误，并且到现在还在出现着错误（参见第25章，25.2节）。

总结：在第1章我们指出，联系最好被当作实体一样看待，在本书中我们一般情况下都这样使用。在第3章中指出，关系模型的一个优点就是用同样的方法——即通过关系变量来代表所有的实体，包括联系。不过，关系的概念在讨论现实世界时确实在直观上看起来是有用的；而且，在13.3~13.5节中所讨论的数据库设计的方法确实对实体和联系之间的区别依赖很大。因此，引入联系这个术语是为了下面几节的讨论。然而，在第13.6节中还会讨论这个问题。

13.3 E/R模型

在13.1节中简要地介绍了最有名的语义建模方法之一——当然也是应用最广泛的一种——称为实体/联系（E/R）方法，它基于Chen在1976年提出的“实体/联系模型”（参见[13.5]），并且在此之后被Chen和许多其它的人用各种方法加以修改（参见[13.13]和[13.4~13.42]）。本章的大部分内容都是针对E/R方法的讨论（然而，应该强调E/R模型远远不是唯一的“扩展”模型——还有许多其它的扩展模型。参见[13.5]、[13.13]、[13.25]、特别是[13.19]中具体介绍了几种其它模型，在[13.22]、[13.31]中给出了有关这个领域的综述）。

E/R模型包括在图13-1中列出的所有类似的语义对象。下面将依次进行介绍。然而，首先应该说明，在文献[13.5]中并不是只介绍了E/R模型本身，还介绍了相应的图表技术（“E/R图”）。在下一节将详细讨论E/R图，在这里只给出一个基于文献[13.5]的图表的例子，如图13-2所示。你将发现它对研究本章所讨论的例子很有帮助。这个例子提供了简单的制造企业的数据（这是在第1章图1-6中讨论的“KnowWare公司”的E/R图的一个扩充版本）。

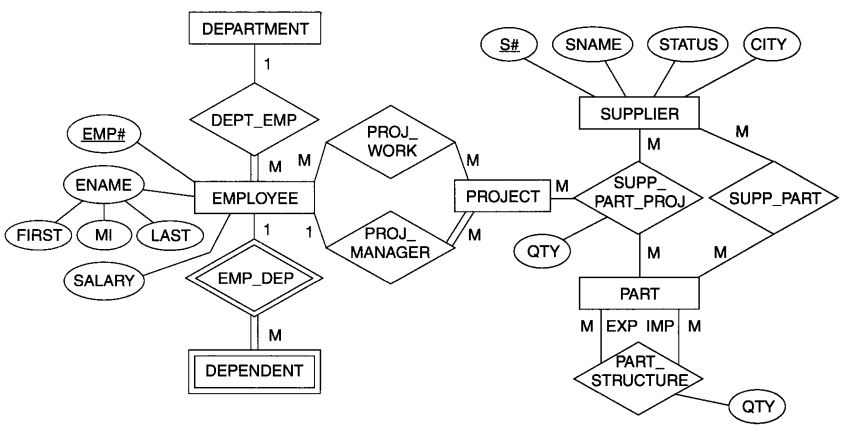

图13-2 实体/联系图（部分示例）

注意：在下面的小节中所讨论的大部分内容对每一个了解关系模型的人来说都相当熟悉。然而，在术语上有某些不同，下面将会看到。

1. 实体

文献[13.5]用“能够被清楚辨识的事物”来定义实体。并将实体划分为常规实体和弱实体。一个弱实体是存在-依赖（existence-dependent）于一些其它实体的实体。就是说，如果其它实体不存在，则它也不存在。例如，根据图 13-2，一个雇员的依赖者（dependent）就是弱实体——当相关的雇员不存在时，它们也不能存在（只考虑数据库）。特别是如果一个给定的雇员被删掉了，则雇员的所有的依赖者都将被删除。与之相反，一个常规实体是非弱实体；例如，一个雇员就是一个常规实体。注意：有些作者用“强实体”而不是用“常规实体”来表示。

2. 特性

实体和联系都具有特性。一个给定类型的所有的实体或联系都有某种共同的特性：例如，所有雇员有一个雇员编号、雇员姓名、工资等特性（注意：在此有意不将“部门号”作为雇员的特性提出。参见下面对联系的讨论）。每一个特性值都是从一个相应的特性集合中取出的（即在关系术语中的域）。而且，特性还可以是：

- 简单的或复合的：例如，复合特性“雇员姓名”可能由简单的特性“first name”、“middle name”和“last name”组成。
- 码（即在某些上下文中是唯一的）：例如，一个依赖者的名字对于一个给定的雇员来说可能是唯一的。
- 单值的或多值的（换句话说，允许重复的组）：所有在图13-2中显示的特性都是单值的，但是如果出现这种情况：例如，一个供应商有几个不同的住地，那么“供应商所在城市”将会是一个多值特性。
- 空缺的（即“不知道的”或“不能适用的”）：这种概念并没有在图13-2中出现，参见第18章的详细讨论。
- 基本的或导出的：例如，对一个给定零件的“总数量”是由该零件的每一笔发货量的总和导出的。图13-2中对此概念也没有示例。

注意：一些作者在E/R模型中用“attribute”而不是“property”。

3. 联系

文献[13.5]定义：联系是“实体之间的一个关联”。例如，在部门和雇员之间有一个联系称为DEPT_EMP，这个联系代表特定的部门雇佣特定的雇员。与实体相同（参见第 1章），在原则上区别联系类型和联系实例是很必要的，但是通常在非正式的讨论中忽略了这样的过程，在下面的讨论中就是这样。

在一个给定联系中的实体称为这个联系中的参与者（participant）。在一个联系中的参与者的数量称为联系的度(degree)（因此要注意：这个术语与在关系模型中的含义有所不同。）

*R*是一个联系类型，*E*是其中的参与者。如果每一个参与者 *E*的实例都至少在*R*的一个实例中出现。那么*R*中的参与者*E*称为全部的(total)，否则它被称为部分的(partial)。例如，如果每一个雇员都必须属于一个部门，那么在关系 DEPT_EMP中的参与者雇员是一个全部的；如果对于一个给定的部门可以根本就没有雇员，那么在联系 DEPT_EMP中部门的参与者就是部分的。

一个E/R联系可以是一对一、一对多（也可以称为多对一）或多对多（为了简单，假定所

有的联系都是二元的，即度数为 2；将这个概念或术语扩展到更高度数的联系当然是很容易的)。现在，如果对关系模型非常熟悉的话，可以认为多对多的情况才是唯一真正的联系，因为这种情况是唯一要求用独立的关系变量来表示 —— 一对一和一对多联系通常可以被一个参与者关系变量中的一个外码来表示。然而，如果一对一和一对多的情况随着时间的推移可以产生或变成多对多的情况，那么就可以将一对一和一对多的情况当作多对多的情况来看待。仅当没有这样的可能性时才能安全地将它们分别对待。当然，有时是不存在这种可能的；例如，一个圆只有一个圆心总是正确的。

4. 实体子类型和超类型

注意：在这一小节中讨论的内容并不包含在最初的 E/R模型中 (文献[13.5])，但是出现于后来的模型中。举例来说，参见Teorey、Yand和Fry[13.41]的理论。

任何给定的实体至少属于一个实体类型，但一个实体可以同时有几个类型。例如，如果一些雇员是程序员（所有的程序员都是雇员），实体类型PROGRAMMER(程序员)是实体类型EMPLOYEE(雇员)的子类型(或者说，实体类型EMPLOYEE是PROGRAMMER的一个超类型)。所有的雇员特性都自动适用于程序员，但反之则不然（例如，程序员可能具有“程序语言的技能”的特性，但一般来讲雇员没有此特性)。同样，程序员自动具有雇员所具有的所有特性，而反之则不对（例如，程序员可能属于一些专业的计算机团体，而一般的雇员则并不如此)。适用于父类的特性和联系可以为子类所继承。

还要注意一些程序员可能是应用程序员而另一些可能是系统程序员；因此称实体类型APPLICATION_PROGRAMMER和SYSTEM_PROGRAMMER都是PROGRAMMER超类型的子类（如此等等)。换句话说，一个实体子类还是一个实体类型，因此可以有它自己的子类。一个给定的实体类型和它的子类、子类的子类，等等……一起组成了一个实体类型层次（参见图13-3)。

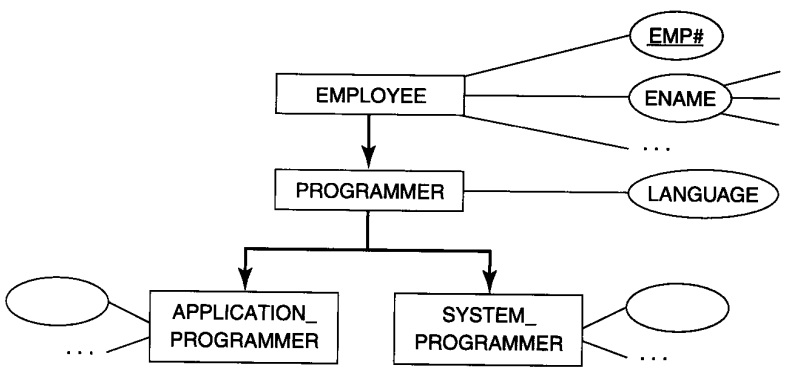

图13-3 实体类型层次的例子

以下几点需要指出：

1) 在第19章会深入地讨论类型层次和类型继承，然而，值得注意的是：在第 19章使用的类型这个术语与第5章中的类型的含义相同，而与本章中讨论的“实体类型”的意义不同。

2) 如果读者对IMS（或某些其它支持层次数据结构的数据库系统）熟悉，可以发现类型

层次与IMS形式的类型层次并不相同。例如，在图 13-3中并没有表明一个 EMPLOYEE 有许多PROGRAMMER与之对应（而如果图中所示的是 IMS的层次结构，则会有这样的对应关系）；相反，对于一个EMPLOYEE的实例至多只有一个相应的PROGRAMMER与之对应，这表明一个雇员只能处于一个程序员的角色。

这就是对E/R模型的主要结构特征的简要描述。现在将注意力转到 E/R图上来。

13.4 E/R图

在前面的一节中已经指出，文献 [13.5]不只介绍了E/R模型本身，还介绍了实体 /联系图的概念（E/R图）。E/R图形成了一种以图形方式表示数据库的逻辑结构的技术。因此，它们提供了一种简单易懂、并可以与任意给定的数据库的设计相交流的方式（“一图胜千言”）。的确，E/R模型作为一种数据库设计方法的普及性更多地归功于 E/R图的存在而不是其它的原因。下面通过在图13-2和图13-3中已给的例子来描述创建一个 E/R图的规则。

注意：正如E/R模型本身，E/R图设计技术也是随着时间的发展而发展的。在这里所描述的版本在某些特定的方面与最初在文献 [13.5]中描述的有所不同。

1. 实体

每一个实体类型都以矩形表示，在矩形中包含所表示的实体类型。对于一个弱实体类型，矩形框是双线的。

例如（参见图13-2）：

- 常规实体：
  DEPARTMENT
  EMPLOYEE
  SUPPLIER
  PART
  PROJECT
- 弱实体：
  DEPENDENT

2. 特性

特性用椭圆显示，其中包含所表示的特性的名字，并且通过实线与相关的实体相连。如果特性是导出的，则椭圆的边用点线或是虚线表示，如果特性是多值的，则椭圆边用双线表示。如果特性是复合的，它的成员特性一样用椭圆来表示，通过实线连接到表示复合特性的椭圆上。码特性用下划线标明。特性对应的值集合不予显示。

例如（参见图13-2）：

- 对于EMPLOYEE：
  EMP#（码）
  ENAME（复合的，包含FIRST、MI和LAST）
  SALARY
- 对于SUPPLIER：
  S#（码）
  SNAME

STATUS
CITY

- 对于SUPP_PART_PROJ:
  QTY
- 对于PART_STRUCTURE:
  QTY

为了节省空间，所有其它的特性都在图13-2中省略了。

3. 联系

每一个联系类型都用菱形框显示，其中包含所表示的联系类型名。如果表示一个弱实体和它所依赖的实体之间的联系，则用双线菱形框。每一个联系的参与者都通过实线与其相关的联系连接；每一个这样的线都标注“1”或“M”来表示这个联系是一对一、多对一，等等。如果参与者是全部的，则用双线连线表示。

例如（参见图13-2）：

- DEPT_EMP(在DEPARTMENT和EMPLOYEE之间的一对多联系)
- EMP_DEP(在EMPLOYEE和一个弱实体DEPENDENT之间的一对多联系)
- PROJ_WORK和PROJ_MANAGER(在EMPLOYEE和PROJECT之间的相互独立的多对多和一对多联系)
- SUPP_PART_PROJ(包括SUPPLIER、PART和PROJECT的多对多对多联系)
- SUPP_PART(在SUPPLIER和PART之间的多对多联系)
- PART_STRUCTURE(在零件和零件之间的多对多的联系)

观察到在最后一个例子中两条从PART到PART_STRUCTURE的两条线通过标注着两个不同的角色名的标签区分（分别为EXP和IMP，即“零件爆炸（explosion）”和“零件闭塞（implosion）”）。PART_STRUCTURE是一个称为递归联系（recursive relationship）的例子。

4. 实体子类型和超类型

实体类型*Y*是实体类型*X*的一个子类型。画一条从*X*的矩形框到*Y*的矩形框的箭头线。这条线代表着一种称为“*isa*联系”的连接（因为每一个*Y*都“是一个(is a)”*X*——即所有*Y*的集合是所有*X*集合的子集）。

例如（参见图13-3）：

- PROGRAMMER是EMPLOYEE的一个子类
- APPLICATION_PROGRAMMER和SYSTEM_PROGRAMMER都是PROGRAMMER的一个子类。

13.5 基于E/R模型的数据库设计

在某种意义上讲，通过在前一节描述的规则构建E/R图就是一个数据库的设计。然而，如果试图将这样的一个设计映射到一个具体形式的DBMS上，就会发现E/R图还是在某些方面不精确，并且它在一些细节方面并未加以考虑（特别是完整性约束的细节方面）。为了阐述这一点，可以考虑一下在将图13-2的设计映射到关系数据库定义上时都包括了什么内容。

1. 常规实体

重复一下，在图13-2中描述的常规实体如下：

DEPARTEMNT
EMPLOYEE
SUPPLILER
PART
PROJECT

每一个常规实体都映射到一个基本关系变量上。因此数据库包括五个基本关系变量：DEPT、EMP、S、P和J，分别与这五个实体类型相对应。而且，这五个基本关系变量中的每一个都有一个候选码——通过属性DEPT#、EMP#、S#、P#和J#表示——它们都与E/R图中确定的“码”相对应。为了定义方便，给每一个关系变量指定一个主码。那么关系变量 DEPT的定义（例如）以如下方式开始：

```
VAR  DEPT  BASE  RELATION
        {DEPT#..., ...}
        PRIMARY   KEY { DEPT#}
```

另外的四个关系变量的定义作为练习留给读者。注意：域或“值的集合”当然也应该被限制。既然在前面已经提到属性值并不包含在 E/R图中，所以省略了对这一方面的细节的讨论。

2. 多对多联系

在例子中多对多（或多对多对多，等等）联系如下：

PROJ_WORK（包括雇员和项目）
SUPP_PART（包括供应商和零件）
SUPP_PART_PROJ（包括供应商、零件和项目）
PART_STRUCTURE(包括零件与零件)

每一个相应的联系也映射到一个基本关系变量上。因此再介绍四种基于这四个联系的基本关系变量。下面看一下SUPP_PART联系。假定这个联系的关系变量是 SP（最常用的发货量关系变量）。为了辨别这个联系中的参与者，先将注意力从这个关系变量的主码问题转移到外码问题上：

```
VAR  SP  BASE  RELATION  SP
   {S#..., P#..., ...}
     ............
   FOREIGN  KEY {S#}  REFERENCES  S
   FOREIGN  KEY {P#}  REFERENCES  P
```

很清楚，与两个参与者（供应商和零件）相对应，关系变量中必须包括两个外码 {S#和P#}，并且这些外码必须参照参与者关系变量 S和P。此外，一组适当的外码规则组合——即一个DELETE规则和一个UPDATE规则——必须是这些具体的外码中的某一个（如果需要根据这些规则来更新内存，请参见第 8章）。在关系变量SP的例子中，指定规则如下（当然具体的规则只是通过图示来说明；要特别注意它们并不能由 E/R模型导出或指定）。

```
VAR  SP  BASE  RELATION  SP
     {S#..., P#..., ...}
       ............
   FOREIGN  KEY  {S#}  REFERENCES  S
          ON  DELETE  RESTRICT
```

```
        ON  UPDATE  CASCADE
FOREIGN  KEY  {P#}  REFERENCES  P
        ON  DELETE  RESTRICT
        ON  UPDATE  CASCADE;
```

对于这个关系变量中的主码又如何呢？一个可能方法是采取参与者-标识（identifying）外码的组合（S#和P#，在SP的例子中）——如果（a）对于每一个联系的实例，这个组合都有一个唯一值（有可能出现这种情况，也有可能不出现，通常情况下是有的）；并且(b)设计者对复合主码无异议。另外的选择是，将一个新的非复合的替代属性，如“发货的数量”，作为主码来使用（参见文献[13.10]和[13.16]）。由于这个例子的原因，选择第一个可能性，给基本关系变量SP的定义加上一条语句：

```
PRIMARY  KEY  {S#,P#}
```

对于PROJ_WORK、PART_STRUCTURE和SUPP_PART_PROJ的联系的讨论作为练习留给读者。

3. 多对一连接

在这个例子中有三个多对一连接：

PROJ_MANAGER(从项目到经理)

DEPT_EMP（从雇员到部门）

EMP_DEP（从依赖者到雇员）

在这三个联系中，最后一个包含了一个弱实体类型（DEPENDENT），而另两个中只包含了常规实体类型。现在来看一下前两种情况。考虑DEPT_EMP的例子。这个例子并不引起任何新的关系变量的引入[⊖]。而事实上只在联系的“多”边上（EMP）的关系变量中引入一个外码，来参照“一”边（DEPT）的关系变量即可。因此有：

```
VAR  EMP  BASE  RELATION
       {EMP#...,DEPT#...,...}
       PRIMARY  KEY {EMP#}
       FOREIGN  KEY {DEPT#}  REFERENCES  DEPT
             ON  DELETE ...
             ON  UPDATE ...;
```

DELETE和UPDATE规则可能性与外码在多对多联系中代表参与者的可能性相同（一般来说）。同样，这不在E/R图中说明。

注意：对于目前的表示，假定一对一联系（实际中，在任何情况下都不常见）与多对一联系的处理方式相同。文献[13.7]中包括了对一对一情况的特殊问题的深入讨论。

4. 弱实体

从一个弱实体类型到它所依赖的实体类型的联系理所当然地是一个多对一联系，在前面的小节中已经介绍了。然而，这个联系的DELETE规则和UPDATE规则必须描述如下：

```
ON  DELETE  CASCADE
ON  UPDATE  CASCADE
```

这些说明合起来共同获取并反映了必要的存在依赖（existence dependence）。下面是一个

⊖ 虽然可能会引入；正如在13.3节提到，有时应该将多对一连接看作是多对多连接。参见文献[18.20]的最后一部分的有关讨论。

例子：

```
VAR  DEPENDENT  BASE  RELATION
   {EMP#..., ...}
   ............
   FOREIGN  KEY {EMP#}  REFERENCES  EMP
            ON  DELETE  CASCADE
            ON  UPDATE  CASCADE;
```

对于这样的一个关系变量来说什么是主码呢？就多对多的情况来说，事实证明可以有几种选择。一种是，如果数据库设计者同意复合的主码，则可以采用外码和 E/R图中的弱实体“码”的组合。另一个选择是，可以引入新的（非复合）替代属性作为主码（参见文献 [13.10] 和[13.16]）。还是由于这个例子的缘故，选择第一种方案，根据基本关系变量 DEPENDENT的定义增加下面的语句：

```
PRIMARY  KEY  {EMP#, DEP_NAME}
```

(DEP_NAME是雇员的依赖者的名字)

5. 特性

在E/R图中显示的每一个特性都映射到一个适当的关系变量的属性 —— 除非特性是多值的，否则通常根据在第11章中讨论的规范化原则为它创建一个新的关系变量（特别是在第 11.6 节）。对一组值的集合可以建立域（虽然值的集合并不可以事先明显地决定）。这些步骤的详细情况是非常简单的，在这里就省略掉了。

6. 实体子类型和超类型

由于图13-2并没有包括任何子类型和超类型，所以参见图 13-3，其中包含了这两种类型。下面看一下实体类型EMPLOYEE和PROGRAMMER。为了简单起见，假定程序员只有一种语言技能（即特性LANG是单值的）。那么[⊖]

- 与往常的方法一样（即已经讨论的方法），超类型EMPLOYEE映射到一个基本关系变量EMP中。
- 子类PROGRAMMER映射到另一个基本关系变量PGMR中，主码与超类型关系变量相同，其它的属性与仅是程序员的雇员的相应属性相同（即在例子中只是 LANG）：

```
VAR  PGMR  BASE  RELATION
     {EMP#..., LANG...}
     PRIMARY  KEY {EMP#};
```

此外，PGMR的主码也是参照关系变量 EMP的外码。因此应将定义扩展为（注意，特别是扩展DELETE和UPDATE规则）：

```
VAR  PGMR  BASE  RELATION
   {EMP#..., LANG...}
   PRIMARY  KEY {EMP#}
   FOREIGN  KEY {EMP#}  REFERENCES  EMP
            ON  DELETE  CASCADE
            ON  UPDATE  CASCADE;
```

⊖ 应特别注意并没有将雇员和程序员映射到一些“超表”和“子表”结构中。这里有一种概念上的困难，或至少是一个陷阱：只是因为在 E/R图中实体类型 *Y*是实体类型 *X*的子类型，这并不表明 *Y*的关系对象就要是 *X*的关系对象的什么“子”类 —— 并且事实上也并不是。详细的讨论参见文献 [13.12]。

• 还需要一个视图EMP_PGMR，它是超类型关系变量和子类型关系变量的连接：

```
VAR  EMP_PGMR  VIEW
     EMP JOIN  PGMR;
```

注意这个连接是（零或一）对一的连接 —— 它是通过一个候选码和一个与之相匹配的外码连接的，并且这个外码本身是一个候选码。因此，大致来说，这个视图包括只是程序员的这些雇员的信息。

下面给出设计如下：

• 可以通过基本关系变量EMP知道所有的雇员的特性（例如，为了恢复的方便）。
• 可以通过基本关系变量PGMR了解只属于程序员的特性。
• 可以通过视图EMP_PGMR了解所有属于程序员的特性。
• 可以通过基本关系变量插入不是程序员的雇员。
• 可以通过视图EMP_PGMR插入所有属于程序员的雇员。
• 可以通过基本关系变量EMP或视图EMP_PGMR删除雇员、程序员或其它。
• 可以通过基本关系变量EMP或视图EMP_PGMR更新所有雇员的特性。
• 可以通过基本关系变量PGMR更新所有程序员的特性。
• 可以通过在基本关系变量 PGMR或视图EMP_PGMR中插入雇员来使一个非程序员变为程序员。
• 可以通过在基本关系变量中删除程序员的方法来使一个雇员从程序员变为非程序员。

对图13-3中的其它实体类型的讨论作为练习留给读者（ APPLICATION_PROGRAMMER和SYSTEM_PROGRAMMER）。

13.6 简单分析

在这一节较深入地分析一下 E/R模型的某些方面。下面的讨论是从文献 [13.8]中对这个问题的详细分析部分中抽取的。另外的分析和注释可以在本章最后一节“ 参考文献 ”中许多参考的评注中找到。

1. E/R模型可以作为关系模型的基础吗

通过从一个不同的角度考虑E/R方法来进行这个讨论。很明显，当Codd第一次提出正式的关系模型的时候，一定是 E/R方法，或一些与这种思想非常接近的想法，支持着 Codd的思维的。如在13.2节中所讨论的，发展一个“ 扩展的 ”模型的总体方法包括四个大的方面，如下：

1) 辨别有用的语义概念；
2) 设计形式化的对象；
3) 设计形式化的完整性规则（“ 元约束 ”）；
4) 设计形式化的操作。

但是这四步对基本的关系模型的设计也是适用的（并且确实对任何正式的数据模型适用），并不只是应用于“ 扩展的 ”模型，如 E/R模型中。换句话说，Codd为了首先构建（形式化的）基本的关系模型，必须在头脑中具有某些（非形式化的）“ 有用的语义概念 ”，并且这些概念必须基本上是这些 E/R模型，或与之相类似的模型。事实上， Codd的著作的确表现出了这种观点。在他的第一篇有关关系模型的论文中（文献 [5.1]的早期版本），可以发现如下内容

“ 一个给定的实体类型的实体集合可以被看作是一个关系，可以称这种关系为实

体类型关系……其它的关系……是在类型之间的……被称为是实体间关系……每一个实体间关系的核心特性[它包括至少两个外码]要么是指相互区别的实体类型，要么是指充当不同角色的一个共同实体类型。”

在这里，Codd清楚地提出了用来代表“实体”和“联系”的关系。但是——在很大程度上——他的观点是：关系是形式化的对象，而且关系模型是一个形式化的系统。 Codd的主要贡献是他发现了对现实世界的某些方面进行了很好描述的形式化模型。

与前面刚刚讨论的相比，实体/联系模型并不是（或者说至少不是主要的）一个形式化模型。它主要包括一组非形式化的概念，这组概念与前面提到的四步中的步骤 1相对应（此外，它所进行的形式化的方面看来并不与基本的关系模型相应的方面有什么重大不同——在下面的一个小节中有对这个问题的详细讨论）。毫无疑问，具有“步骤 1”概念的思想将会对进行数据库设计的处理工作很有帮助，但是如果没有形式化的对象和步骤 2、3，数据库设计就不能完成，其它的一些任务没有步骤 4的形式化的操作也不能完成。

请注意前面的讨论并不是要表明 E/R模型没有用，相反它是有用的。但这并不是全部。而且奇怪的是最早发表的非形式化的 E/R模型的描述是在最早的形式化关系模型的描述发表几年之后，因而存在这样的假定（正如我们已经谈过的），即，后者起初是建立在某些类似 E/R思想的基础上的。

2. E/R模型是一个数据模型吗

由前面的讨论，甚至并不能判断 E/R“模型”是否真是一个数据模型，至少在本书中所用的这个术语本身的意义上来讲（即作为正式的系统应包括结构、完整性和操作等方面）。当然，“E/R建模”这个术语经常用来表示决定数据库结构（只是数据库的结构）的过程，虽然在第 13.3~13.5节中也简单地讨论了特定的完整性因素问题（主要是与主码和外码相关的问题）㊀。然而，仔细地阅读文献 [13.5]会发现，E/R模型确实是一个数据模型，但在本质上只是基础的关系模型顶上的一个薄层（正如一些人所述，它当然并不是替代关系模型的一个候选模型），下面证明如下：

- 首先基本的 E/R数据对象——这就是说，基本的正规对象，与不正规的对象“实体”、“联系”，等等相对立——是一个*n*元关系。
- E/R操作是基本的关系代数的运算符（实际上，文献 [13.5]在这一点上解释得并不是非常清楚，但提出一组这样的操作显然没有那些关系代数的操作全面；例如，显然在这里没有并集操作和明显的连接操作）。
- 在完整性方面，E/R模型有一些关系模型没有的功能：E/R模型包括一组内置的完整性规则，与在本书中所讨论的一些而不是全部的外码相对应。因此，“纯”关系系统需要用户显式地构建特定的外码规则，而 E/R系统只要求用户标明一个给定的关系变量表示某些联系，这样，一些外码规则就可以很好地理解。

3. 实体与关系

本书中已经几次指出，“关系”最好只被当作一种特殊类型的实体。而相反地，区分这两

㊀ 这当然是一个主要的弱点，E/R模型除了几个特殊的情况(公认是重要的情况)之外是完全不能处理完整性约束或“企业规则”的，这里引用一句经典的话：“说明性(declarative)的规则太复杂，在企业模型中很难遵循，必须由分析员/开发人员单独定义”[13.27]。有一种观点认为数据库设计应该是一个限制应用约束的过程(参见文献[8.18~9.19])和文献[13.17~13.19]。

个概念是E/R方法的一个必要条件。作者的观点是任何坚持做这样区分的方法一定是有严重缺陷的，因为（在13.2节中提到）同一个对象可以被一些用户看作是实体而为另一些用户当作是联系。下面看一个关于婚姻的例子：

- 从一个角度看，结婚是两个人之间的联系（一个查询例子：“谁在1975年和伊丽沙白泰勒结婚？”）。
- 从另一个角度看，结婚对每一个实体本身都是同样的（查询例子：“自从4月份以来在这个教堂进行了多少次婚礼？”）。

如果设计方法强调区分实体和联系，那么（最好的情况是）这两个解释将会被不对称地对待（即“实体”查询和“联系”查询会采用不同的形式）；在最坏的情况下，一种解释方式根本不支持另一种解释方式（即某一类查询将无法进行）。

有关这一点更深层的说明见[13.17]中关于E/R方法的介绍中论述：

> 最初常见的方法是，在概念模式设计中用属性[特指外码]来表示联系，然后在设计进行中将这些属性转换为联系，使这些属性更好理解。

但是当一个属性变成一个外码时会发生什么情况呢？——即数据库在已经存在了一段时间后进化——在这种情况下会发生什么呢？如果因此而作出逻辑结论，那么数据库设计就应该只包括联系而根本不包括属性（事实上，这种设计确实有一些好处。参见文献[13.18]的注释）。

4. 最后一个分析

除了在本章中描述的E/R建模这个特定的语义建模方案之外还有很多其它的方案。然而，这些方案绝大部分都具有一种很强的相似性；特别是它们都可以被刻画为表示特定的外码约束和几个其它的微小方面的简单的图形符号。这样的图形表示法在“大图（big picture）”方式中当然有用，但是它们太单纯，不能应付整个的设计工作[⊖]。特别是如前面已经提到的，它们一般不能处理通常的完整性约束。例如，如何在一个E/R图中表示一个连接依赖呢？

13.7 小结

在本章中简单地介绍了语义建模这个概念。共包括四个步骤，其中第一步是非正规的，而其它的都是正规的：

1) 区别语义概念；
2) 设计相应的符号对象；
3) 设计相应的完整性规则（“后约束”）；
4) 设计相应的操作符。

一些有用的语义概念是实体、特性、联系和子类型。注意：我们还强调一点，(a) 在语义建模层（不正规的）和其系统层(正规的)之间很可能存在术语冲突，(b) 这种术语冲突可以引起混乱！

语义建模研究的最终目的是使数据库系统更智能化。一个更直接的目标是为系统地解决数据库设计中的问题提供一个基础。我们描述了一种特定的“语义”模型的应用，即Chen的

⊖ 很可悲的是，在这个行业中，简单的甚至过于简单的答案是很受欢迎的。在这个问题上我们同意爱因斯坦的话“每件事都应该做得尽可能的简单——不能再简单为止”。

实体/联系模型（E/R模型）。

与前文所述相联系，需要重申的是：最初的 E/R论文[13.5]中确实包含两个互相区别的多少有些独立的提议：即提出了 E/R模型本身以及E/R图技术。正如在第13.4节中所论述的，E/R模型的普遍性可以归功于图表技术的存在而并非其它的原因。但是只为了用图没必要采用所有的模型思想；将 E/R图作为任何设计方法 —— 例如，在文献 [13.6]的一个基于 RM/T的方法 —— 的基础都是可能的。有许多讨论将 E/R模型和一些其它方法的相符性作为数据库设计的基础，在这些讨论中通常没考虑到这一点。

下面将语义建模的思想（特别是E/R模型）和在第11、12章中讨论的规范化理论作一比较。规范化理论是将大关系变量分解为小的关系变量；它假定初始有数量较少的大关系变量，通过规范化理论将这些初始输入的大的关系变量变成小的关系变量输出 —— 即将大的关系变量映射为小的关系变量（当然，在这里只是非常简略地提一提）。但是规范化理论对于首先如何得到的那些大的关系变量却没有提到。然而，像本章所讨论的自上而下的设计方法确切地揭示了这个问题；它们将现实世界映射到这些大的关系变量中。换句话说，这两个方法（自上而下的方法和规范化方法）互为补充。因此总体设计过程如下：

1) 用E/R方法（或类似的方法）产生“大的”具有常规实体、弱实体等的关系变量；

2) 利用进一步规范化的思想将这些“大的”关系变量分解为“小”的关系变量。

然而，从本章讨论的内容上说，语义建模还不像在 11章和第12章中讨论的进一步规范理论那样显得严格分明。原因是（正如在本书这一部分的引言中所谈到的）数据库设计是一个非常主观的活动，并不是非常客观的；可以作为处理这些问题的基本原理相对来说是比较少的（当然确实存在这样的一些原理，基本上前两章所讨论的内容就是一些这样的基本原理）。虽然本章的方法确实在实际问题中是有效的，但它们仍然只能被看作经验准则。

最后还有一点值得说明。虽然整个方面还有些主观，但在一些具体的方面语义建模思想是非常有用的 —— 即在数据字典中。数据字典可以在某些方面被看作是“数据库设计者的数据库”；在这个数据库中，数据库设计的行为与其它的操作一起被记录（文献 [13.2]）。语义建模的研究因此在字典系统的设计中是很有用的，因为它辨别了字典本身需要支持和“理解”的一类对象 —— 例如，实体目录（entity category）（例如E/R模型中的常规实体和弱实体）、完整性约束（如在 E/R模型中一联系的全部与部分参与者的表述）、实体超类型和子类型，等等。

练习

13.1 谈谈对“语义建模”的理解。

13.2 试述在E/R模型中定义一个“扩展”模型所包括的四个步骤。

13.3 定义下面的E/R术语：

实体，联系，继承，父类型和子类型

码属性，类型层次，特性，值的集合，常规

实体，弱实体

13.4 举出下面几种情况的例子：

a) 一个参与者是弱实体的多对多联系。

b) 一个参与者是另一个联系的多对多联系。

c) 有一个子类型的多对多联系。

d) 一个有父类型中不存在的弱实体的子类型。

13.5 根据第8章中练习8.10画一个教育数据库的E/R图。

13.6 根据第11章中练习11.3的公司员工数据库画一个E/R图。应用这个图导出一组适当的基本关系变量定义。

13.7 根据第11章中练习11.4的数据库画一个E/R图。用这个E/R图来导出一组适当的基本关系变量定义。

13.8 根据第12章的练习12.3中的销售数据库画一个E/R图。利用这个E/R图导出一组适当的关系变量定义。

13.9 根据第12章的练习12.5中修改后的销售数据库来画一个E/R图。利用这个E/R图导出一组适当的关系变量定义。

参考文献和简介

下面列出的参考文献之所以显得长了一些，是由于在数据库界近年来确实提出了大量的数据库设计方法，有来自工业界的也有来自学术界的。在这个领域很少有一致的意见；本章中所介绍的E/R模型当然是应用最广泛的一种方法，但是并不是所有人都接受它（或喜欢它）。事实上，应用最广泛的方法并不一定是最好的方法。其实许多商业产品已远远超出了只提供数据库设计工具的范畴，它们已可以做到去生成整个应用——除了特定的数据库定义（模式）外，还有前端屏幕设计、应用逻辑、触发过程，等等。

其它与本章内容相关的文献包括ISO有关概念模式的报告[2.3]；Kent的《Data and Reality》[2.4]；Ross的关于企业规则的著作[8.18~8.19]。

13.1 J. R. Abrial: “ Data Semantics,” in J. W. Klimble and K. L. Koffeman (eds.), *Data Base Managenlent*. Amsterdarn, Netherlands: North-Holland / New York, N.Y.: Elsevier Science (1974).

在语义建模领域的一个早期的提议。下面引用的话很好地描述了这篇论文的大体格调：“对读者的提示：如果要找语义这个术语的定义，请就此打住，因为本文中没有这样的定义”。

13.2 Philip A. Bernstein: “ The Repository: A Modern Vision,” *Database Programming and Design* 9, No. 12 (Decernber 1996).

在写这篇论文的时候好像正在面临由术语信息中心（repository）取代术语数据字典的趋势。一个信息中心系统是特定应用于元数据(metadata)管理的DBMS——元数据并不只是用于DBMS而是用于各种类型的软件工具，引用Bernstein的话说：“是软件设计、开发和配置的工具，也是与工程内容相关的管理工具，比如对电子设计、机械设计、网址和许多其它方面的正式文档的管理等”。这篇论文是有关信息中心概念的一个辅导。

13.3 Michael Blaha and William Premerlani: *Obiect-Oriented Modeling and Design for Database Applications*. Upper Saddle River, N.J.: Prentice-Hall (1998).

详细描述了一个称为对象建模技术（OMT）的设计方法。OMT可以看作是E/R模型的一个变种——它的对象是基本的E/R实体——但是它远不止限于数据库设计范畴。参见文献[13.32]的注释。

13.4 Grady Booch: *Object-Oriented Design with Applications*. Redwood City, Calif.: Benjamin/Cummings (1991).

参见文献[13.32]的注释。

13.5 Peter Pin-Shan Chen: " The Entity-Relationship Model – Toward a Unified View of Data," *ACM TODS 1*, No. I (March 1976). Republished in Michael Stonebraker (ed.), *Readings in Database Systems*. San Mateo, Calif.: Morgan Kaufmann (1988).

这篇论文介绍了E/R模型和E/R图。在本章中已经介绍过，这个模型是随着时间推移不断地修改和完善的；在这第一篇论文中的提出的解释和定义肯定不十分精确，所以这样的完善是完全必要的（一个对E/R模型的批评是在此模型中的术语并没有一个单一而完善的含义，相反却可以用很多方式来解释。当然，整个数据库领域都受不精确和术语冲突所苦恼，但是E/R建模这个具体的领域的情况却比大多数其它的领域都差），表现如下：

- 正如在13.3节中所介绍的，一个实体定义为" 一个可以被清楚辨识的事物 "，一个联系定义为" 与其它实体的连接 "。这样，第一个问题出现了：联系是实体吗？一个联系也是" 可以被清楚辨识的事物 "。但是这篇论文中下面的几节却保留了术语" 实体 "来表示某些不是联系的东西。假定后面的是有意的解释，那么为什么有术语" 实体/联系 "模型出现呢？但是这篇论文中没有给出清晰的解释。
- 实体和联系可以有属性（在本章中所应用的是" 特性 "一词）。这篇论文在这个术语的含义上又出现了自相矛盾的情况 —— 首先它定义了一个属性，它是一个非主码也非主码的任何组成部分的特性（与关系的定义相对来说），但其后它在关系的意义上又用到这个术语。
- 一个联系的主码被假定为辨别了联系中的实体的外码的组合（但其中并没有用到外码这个术语）。这个假定只适用于多对多的联系，而且并非总能适用。例如，对于一个关系变量SPD{S#,P#,DATE,QTY}，它代表了在某些天中某些供应商提供的某些零件的数量；假定同意供应商可以多次运送同样的零件，但是不在同一天。那么主码（或者至少是单独的候选码）是{S#,P#,DATE}的组合；然而可以将供应商和零件作为实体来看待，但日期却不能。

13.6 E. F. Codd: "Extending the Database Relational Model to Capture More Meaning, " *ACM TODS* 4, No. 4 (December 1979).

在这篇论文中，Codd介绍了一个" 扩展 "的关系模型版本，他称之为RM/T。RM/T论述了与E/R模型同样的一些问题，但是它定义得更详细。这两种模型的一些直接的不同如下：首先，RM/T对于实体和联系并没有作任何不必要的区分（一个联系只是作为一个特殊类型的实体类看待）。第二，RM/T较之E/R模型的结构和完整性方面更庞大并且定义得更精确。第三，RM/T在基本的关系模型的操作符基础上建立它自己的特殊的操作符（虽然许多附加的工作在后面还需完善）。

在大体结构上，RM/T模型的内容如下：

- 首先，实体（包括" 联系 "）是由*E*关系和*P*关系代表的[⊖]，这两种关系都是一般的*n*元关系的特殊形式。*E*关系用来记录存在的实体，*P*关系用来记录这些实体的特性。
- 第二，一组不同的联系可以在实体中存在 —— 例如，实体类型*A*和*B*可以用一个关联

⊖ 或者更恰当地说，是*E*关系变量和*P*关系变量。

（association）（RM/T中的多对多联系的术语）被链在一起，或实体类型 *Y*可以是实体类型*X*的子类型。RM/T包括一个正式的目录结构，通过这样的结构系统就可以了解这些联系；因此，系统能够保证由这些联系所蕴涵的各种完整性约束。

- 第三，提供了许多更高级别的操作符来方便不同的 RM/T对象的处理（*E*关系、*P*关系、目录关系，等等）。

与E/R模型相似，RM/T模型也包含了图13-1中列出的所有的概念（实体、特性、联系、子类型）。它具体地提供了一种实体分类模式（这种模式在很大程度上形成了整个模型的最重要的方面——或者说，最直接可见的方面），根据这种模式，实体被划分为三类，即内核、特征和连接：

- 内核(kernel)：内核实体是指有独立存在性（ independent existence）的实体：它们体现的是“数据库究竟是什么？”。换句话说，内核是既非特征也非关联的实体（参见下面）。
- 特征(characteristic)：一个特征实体是以描述和表现一些其它实体属性为主要目的的实体。特征是它们所描述的实体上的存在依赖 (existence-dependent)。所描述的实体可以是内核，特征和关联。
- 关联：一个关联实体是在两个或多个其它实体之间的多对多联系（或多对多对多，等），关联的实体可以是内核、特征和关联。

另外：

- 实体（不管属于哪一类）可以有特性。
- 特别地，任何实体（不管其属于哪一类）都可以指派（ designate）一些其它的相关特性。一个指派代表两个实体之间的多对一联系。注意：在最初的论文 [13.6]中没有讨论指派，这是后来加入的。
- 支持实体子类型和超类型。如果 *Y*是*X*的一个子类型，那么 *Y*是一个内核，特征还是关联，依赖于X是一个内核、特征还是关联。

可以将前面的概念与E/R模型做一比较（大致上）如下：一个内核与 E/R“常规实体”相当，一个属性与 E/R“弱实体”相当，一个关联与 E/R“联系”相当（只是多对多情况）。

注意：另一个偶尔会遇到的术语主域（primary domain）也是在这篇论文中第一次定义的。一个主域是至少有一个单属性主码的域。在供应商和零件的例子中，主域是S#和P#。

除了上面简单介绍的方面外，RM/T还包括对以下内容的支持：(a)替代(surrogates)(参见文献[13.16])，(b)时间维（参见第22章），(c）各种数据聚集（参见文献[13.35~13.36]）。

13.7 C. J. Date: “A Note on One-to-One Relationships,” in *Relational Daabase Writings 1985-1989*.Reading, Mass.: Addison-Wesley (1990).

关于一对一联系问题的一个广泛的讨论，证明了一对一问题比最初看来要复杂得多。

13.8 C. J. Date: “Entity/Relationship Modeling and the Relational Model,” in C. J. Date and Hugh Darwen, *Relational Database Writings 1989-1991*. Reading, Mass.: Addison-Wesley (1992).

13.9 C. J. Date: “Don’t Encode Information into Primary Keys!”, in C. J. Date and Hugh Darwen, *Relational Database Writings 1989-1991*. Reading. Mass.: Addison-Wesley (1992).

这篇论文提出了一系列有关“智能码 (intelligent key)”的非正式的讨论。关于外码问题参见文献[13.10]中的相关介绍。

13.10 C. J. Date: “ Composite Keys,” in C. J. Date and Hugh Darwen, *Relational Database Writings* 1989-1991. Reading. Mass.: Addison-Wesley (1992).

引自摘要：“总结了关于是否支持在关系数据库设计中包含的复合（码）的问题，并且给出……一些建议”。而且，这篇文章中指出了替代码（文献[13.16]）的问题。

13.11 C. J. Date: “ A Database Design Dilemma? ”, on the *DBP&D website www.dbpd.cotn* (January 1999). See also Appendix B of reference [3.3].

从表面判断，一个给定的实体类型，如雇员，在关系系统中既可以通过雇员类型（即一个域）表示，也可以通过雇员关系变量表示。这篇短文给出了如何在两者中作出选择。

13.12 C. J. Date: “ Subtables and Supertables ” (in two parts), on the *DBP&D website www.dbpd.com* (to appear late 2000 or early 2001). See also Appendix D of reference [3.3].

通常认为实体类型继承应该在关系上下文中通过一种“子表和父表”——实体子类型映射到一个“子表”，实体父类型映射到一个“父表”——的方式来处理。例如，SQL3就支持这样一种方法，某些产品也支持。这篇论文对此持反对意见。

13.13 Rarnez Elmasri and Sharnkant B. Navathe: *Fundamentals of Database Systems* (2nd edition). Redwood City. Calif.: Benjamlii/Cummings (1994).

这本关于数据库管理的教材中用整整两章（超过 25页）讨论了在数据库设计中如何使用E/R技术。

13.14 David W. Enibley: *Object Database Development: Concepts and Principles*. Reading, Mass.: Addison-Wesley (1998).

提供了一种基于OSM（Object-oriented Systems Model, 面向对象的系统模型）的设计方法。OSM的某些部分很像ORM[13.17~13.19]。

13.15 Candace C. Fleming and Barbara von Hallé: *Handbook of Relational Database Design*. Reading, Mass.: Addison-Wesley (1989).

针对关系系统中数据库设计的一个实例指南，包括基于 IBM的DB2产品和Teradata（现在是NCR）DBC/1012产品的具体的例子。在这本书中逻辑设计和物理设计的内容都提到了——虽然这本书应用“逻辑设计”的术语来表示“关系设计”，用“关系设计”这个术语包括“物理设计”中的一些方面。

13.16 P. Hall, J. Owlett. and S. J. P. Todd: “ Relations and Entities,” in G. M. Nijssen (ed.), *Modelling in Data Base Management Systems*. Amsterdam, Netherlands: North-Holland / New York, N.Y.: Elsevier Science (1975).

这是第一篇详细论述替代码概念的论文（这个概念后来用于 RM/T中[13.6]）。替代码是通常意义上的关系中的码，但是有如下几个特殊的特征：

- 它们通常只包括一个属性。
- 它们的值只作为它们代表的实体的替代。换句话说，这些值只是表明相应的实体存在——它们并不带有其它任何信息或含义。
- 当一个新实体插入到数据库中，就会被赋予一个从未用过的替代码的值，并且这个

值也不会再用，甚至当这个实体已经被删除后也不能用。

最好的情况是，替代码是系统生成的，但是，无论是系统生成的还是用户给出的，都与替代码这个基本思想无关。

值得强调的是替代码并非（如一些作者所认为的）是与“元组 ID”相同。第一，明确地说，元组ID标识了元组而替代码表示了实体，并且在元组和实体之间并没有一对一的联系（特别是对导出元组的ID）。而且，元组ID有性能上的含义而替代码没有；通过元组ID来访问元组通常被认为是很快的（假定元组——至少是在基本关系中的元组——直接映射了物理存储，像现在的许多产品一样）。并且元组ID通常是对用户隐藏的，而替代码不能对用户隐藏（由于信息原理）；换句话说，不可能像一个属性值一样来存储一个元组ID，但可以这样存储一个替代码。

简单一句话：替代码是一个逻辑概念；而元组 ID是一个物理概念。

13.17 Terry Halpin: *Conceptual Schema and Relational Database Design* (2nd edition). Sydney, Australia: Prentice-Hall of Australia Pty., Ltd. (1995)

一个对ORM的详细介绍（参见下面两文献的介绍）。

13.18 Terry Halpin: “Business Rules and Object-Role Modeling,” *DBP&D* 9, No. 10 (October 1996).

一个对象-角色建模（object-role modeling, ORM[13.17]）的非常好的介绍。Halpin开始其研究，并认为“[不像]E/R建模，存在很多方言，ORM只有很少的方言，并且区别很小”（注意：其中的一个方言是 NIAM[13.29]）。ORM还以基于事实建模著称，因为设计者所做的是写下——或者是用自然语言，或者是通过一种特殊的图符号——一系列基本事实（或是事实类型（fact type）），这些事实在一起刻画了要被建模的企业的特征。这样的事实类型的例子如下：

- 每一个雇员至多有一个雇员姓名 Empname.
- 每一个雇员至多向一个雇员汇报。
- 如果雇员*e1*向雇员*e2*汇报，那么*e2*不能向雇员*e1*汇报。
- 没有雇员可以既指导又进入同一个项目。

正如所见，事实类型其实就是谓词或企业规则；正如 Halpin的论文题目所表明的，ORM确实是数据库设计方法的精华，它通常被“企业规则”的倡导者和作者所喜欢。通常，在联系中对象所扮演的角色是由事实指定的（因此有“对象-角色建模”的名字）。注意（a）在这里“对象”表示实体，而不是在这本书第六部分中描述的特殊意义上的对象；（b）联系并不必是二元的。然而，事实是基本的——它们不能分解为更小的事实。注意：数据库只应该包含在概念层的基本（或不可分解的）事实的观点是由 Hall、Owlett和Todd[13.16]更早提出的。

观察到ORM没有“属性”概念。因此，在这篇文章中显示（参见文献[13.19]的注释），ORM设计在概念上比 E/R模型设计的相应部分更简单并且更强。然而，属性可以并且确实出现在由ORM设计所（自动）产生的E/R模型或SQL中。

ORM还强调了“样本事实（ sample fact）”（即，样本事实实例——称为命题（proposition））的应用，即作为允许终端用户检查设计合法性的方法。基于事实的建模的方法是易懂的，而E/R建模则要差许多。

存在许多描述一个企业的逻辑上相同的方法，因此有许多逻辑上相同的方案。ORM包含一组转换规则，这种规则允许逻辑上相同的方案互相转换，所以一个ORM工具在设计者指定的设计上可以进行一些优化。如前面提到的，它还可以从一个ORM方案产生E/R或SQL方案，并且它还能相反地从一个存在的E/R或SQL方案产生一个ORM方案。基于DBMS目标，一个产生的SQL方案可以包含（SQL-92标准）定义的约束或那些可以通过存储或触发过程执行的约束。顺便提及，考虑这些约束时注意——并不像E/R建模——ORM定义时包含“一个表现约束的丰富的语言”。（然而，Halpin在文献[13.19]中确实承认，并不是所有的企业规则都可以用ORM图形符号表现出来——在这种情况下，文本描述也还是需要的）。

最后，一个ORM方法可以被看作是一个高级的抽象的数据库视图（事实上，它是相当接近于一个纯粹的规则化的关系视图）。因此，它可以直接查询。参见下面文献[13.19]中的注释。

13.19 Terry Halpin: “Conceptual Queries,” *Data Base Newsletter* 26, No.3 (March/ April 1998).

引自摘要：“在关系语言中构造非平凡查询如SQL或QBE，可以证明会使用户畏缩。ConQuer，一个基于对象-角色建模（ORM）的新的概念查询语言可以方便用户以一种易于理解的方式构造查询……”该文指出了在构造查询和企业规则上，该语言要优于传统查询语言。

在其它讨论中，这篇论文还讨论了一个如下的ConQuer查询：

✓Employee
  +– drives Car
  +– works for ✓Branch

（“找出驾驶轿车的雇员及其所在部门”）。如果雇员可以驾驶任何数量的小轿车但只能在一个部门工作，其SQL设计就要包括两个表，产生的SQL代码如下：

```
SELECT DISTINCT X1.EMP#, X1.BRANCH#
FROM   EMPLOYEE AS X1, DRIVES AS X2
WHERE  X1.EMP# = X2.EMP# ;
```

现在假定对于每一个员工同时在多个部门工作是可能的。那么SQL设计就要变化为包括三个表而不是两个，产生的SQL代码也将变化为：

```
SELECT DISTINCT X1.EMP#, X3.BRANCH#
FROM   EMPLOYEE AS X1, DRIVES AS X2, WORKS_FOR AS X3
WHERE  X1.EMP# = X2.EMP# AND X1.EMP# = X3.EMP# ;
```

然而，ConQuer查询保持不变。

正如上面的例子所示，像ConQuer这样的语言可以看作是提供一个特别强的逻辑数据独立性形式。为了解释这种说法，首先需要细化ANSI/SPARC体系结构[2.1~2.2]。在第2章中提到逻辑数据独立性表示对于概念模式的变化的独立性——但是，前面的例子的总体观点是概念模式的变化并没有发生！问题是现在的SQL产品并不能很好地支持一种概念模式，而只是支持SQL模式。SQL模式可以被看作是存在于真正的概念层和内部或说物理层之间的中间层。如果一个ORM工具允许定义一个真正的概念模式，并且将这个模式映射到一个SQL模式上，那么ConQuer就可以提供SQL模式的独立性（当然是通过对映射做适当的改变）。

在这篇论文中并没有清楚说明ConQuer的表现力究竟有什么限制。Halpin并不直接谈

论这个问题；而是说：“这种语言应该很完美地允许根据应用来形成相应的问题；在实践中，较之稍差一些的语言也是可以接受的”。它还提到ConQuer的“最强大的特征……是其能够执行任意复杂的相关性查询”。下面是一个例子：

✓Employee1
+–lives in City1
+– was born in Country1
+– supervises Employee2
+– lives in City1
+– was born in Country2 <> Country1

(“找出监督一个与其住在同一个城市但不在一个城市出生的雇员的雇员信息”)。正如Halpin所说的“试着将查询用SQL表示！”。

最后，根据ConQuer 和企业规则的观点，Halpin指出：“虽然ORM的图形符号可以较之[E/R方法]描述更多的企业规则，它还是需要文本语言的补充 [来表示特定的约束]。将ConQuer应用于这个目的的研究正在进行。

13.20 M. M. Hammer and D. J. McLeod: “The Semantic Date Model: A Modelling Mechanism for Database Applications, ” Proc. 1978 ACM SIGMOD Int. Conf. on Management of Data, Austin, Texas (May/June 1978).

语义数据模型（SDM）代表另一种数据库设计形式。与E/R模型相似，它关注了结构和（某种程度上）完整性方面的问题，并且在操作方面言之很少，甚或什么都没有说。参见文献[13.21]和[13.24]。

13.21 Michael Hammer and Dennis McLeod: “ Database Description with SDM: A Semantic Database Model,” *ACM TODS* 6, No. 3 (September 198 1).

参见文献[12.20]的注释。

13.22 Richard Hull and Roger King: “ Semantic Database Modeling: Survey, Applications, and Research Issues,” *ACM Comp. Surv.* 19, No. 3 (September 1987).

一个全面的对在20世纪80年代末语义建模领域及相关情况的综述。这篇文章是对此问题开始一个更深入的调查和研究语义建模行为有关情况的很好的起点。参见文献[13.31].

13.23 Ivar Jacobson *et ad.: Obiect-Oriented Software Engineering* (revised printing). Reading, Mass.: Addison-Wesley (1994).

描述一种称为面向对象软件工程（OOSE）的设计方法。与OMT[13.3]相似，OOSE的数据库部分至少可以看作是E/R模型的一个变体（与OMT相同，OOSE的对象是基本的E/R实体）。下面的引述是值得注意的：“现在在工业中应用的所有方法，对于信息技术系统发展来说，都是基于系统的函数和 /或数据-驱动的分解。这些方法在许多方面和面向对象方法不同，在后一种方法中，数据和函数是有很高的结合度的”。看起来在这里Jacobson关注了在对象和数据库思想之间的重要的不符合性。数据库 —— 至少是普遍意义上的共享数据库，这种数据库是数据库领域所关注的主要焦点 —— 被假定是与“函数”分离的；假定它们是与使用它们的应用系统分离设计的。因此，在对象领域里，“数据库”的含义是指一个特定的应用的数据库，并不是一个共享和普遍意义上的数据库。
还可以参见（a）参考[13.32]的注释，（b）第24章对象数据库的讨论。

13.24 D. Jauannathan *et td*.: " SIM: A Database System Based on the Semantic Data Model," Proc.1988 ACM SIGMOD Int. Conf. on Management of Data, Chicago , I11. (June 1988).

描述了一个基于与 Hammer和McLeod在文献[13.20]中提出的语义数据模型相似的"一个语义数据模型"的商业化 DBMS产品。

13.25 Warren Keuffel: " Battle of the Modeling Techniques: A Look at the Three Most Popular Modeling Notations for Distilling the Essence of Data," *DBMS* 9, No. 9 (August 1996).

"三个最普遍的符号"是 E/R建模， Nijssen的自然语言信息分析方法，即NIAM[13.29],和语义对象建模，即 SOM。Keuffel称E/R建模是其它两种方法的"祖父"，但是他批评了这种方法缺少正式的根基；他说，实体、联系和属性（即特性）都是"没有参考它们是怎么发现而描述的"。NIAM更严格一些；当严格遵守其设计规则时，所得出的概念设计才比用其它方法设计的东西"具有更多的完整性"，虽然"一些开发者发现NIAM的严格设计太闭塞"！对于 SOM，它与"E/R模型非常相似……它们有相同的对于实体、属性和联系的模糊定义"；然而，它与 E/R模型也有不同，它支持组属性（即重复组），组属性允许一个"对象"（即实体）包括其它对象（E/R建模允许实体包括属性的重复组，但不允许包含其它实体）。

13.26 Heikki Mannila and Kari-Jouko Räihä: *The Design of Relational Databases*. Reading, Mass.: Addison-Wesley (1992).

引用该书的前言，是"一本适合研究生水平的教材，是关系数据库设计的参考书"。它一方面包含依赖理论和规范化理论，另一方面包含了 E/R方法，每种情况都是从一个正规的角度来描述的。下面的这些章节列出了这本书的大概范围：

- 设计原理
- 完整性约束和依赖
- 关系模式的特性
- 依赖公理
- 有关设计问题的算法
- E/R图和关系数据库模式之间的映射
- 模式转换
- 设计中的实例数据库

作者对该书介绍的技术，用一种称为 Design By Example的工具加以实现。

13.27 Terry Moriarty: *Enterprise View* (regular column), *DBP&D 10*, No. 8 (August 1997).

描述了一种企业应用设计和一种称为 Usoft的开发工具，它们允许用一种像 SQL一样的语法来定义企业规则，并且应用这些规则产生应用（包括数据库定义）。

13.28 G. M. Nijssen, D. J. Duke, and S. M. Twine: " The Entity-Relationship Data Model Considered Harmful," Proc. 6th Symposium on Empirical Foundations of Information and Software Sciences, Atlanta, Ga. (October 1988).

"E/R模型被认为是有害的么？"看起来确实有许多需要回答，包括：

- 类型和关系变量引起的混乱（参见第 25章中关于第一个大错误的讨论）；
- 关于"子表和父表"的奇怪的事情 [13.12]；
- 一个散布很广的对数据库相对准则的估价的失败（参见第 9章）；

- 实体和联系本身引起的混乱，在前一章论述。

文献[13.28]中提出许多疑问。具体地说，它声称E/R模型：

- 提出太多提供数据结构的重叠的方法，因此使设计过程不适当；
- 没有指出如何选择不同的表示法，并且事实上如果环境变化，可以使现有的设计发生不必要的变化；
- 很少支持保持数据完整性的方法，因此不能完成设计过程的某些方面（“约束可以用一个更普遍的符号[如]谓词逻辑来形式化地表示，但据此认为这是一个从数据模型中可以省略约束的合理理由就不太合适了，这好比一种程序设计语言，由于它自身不能表达所有的功能，所以需要用户调用汇编语言来完成，却要称这种程序语言是完善的语言一样”）。
- 与普遍的观点相反，它并不是一个最终用户和数据库专业人员之间交流的很好的工具；
- 违反了概念化原则：“一个概念模式应该……只包括与概念相关的方面，不论是动态的还是静态的，排除所有无关的内容，如数据表示（外部的或内部的）、物理数据的组织和存取以及消息格式、数据结构等特定外部用户表示，等等”[2.3]。事实上，作者暗示E/R模型是老的CODASYL网络模型（参见第1章）的一个“再生”。“对实现结构的强烈偏见是否成为E/R模型在专业领域得到这么广泛的认同的主要原因呢？”。

这篇文章还详细地分析了E/R模型的许多弱点。然后提出了一个可选方法NIAM[13.29]。它特别强调了NIAM并不包含E/R模型中所不必要包含的属性和联系的区别。

13.29 T. W. Olle, H. G. Sol. and A. A. Verrijn-Stuart (eds.): *Information Systems Design Methodologies: A Comparative Review.* Amsterdam, Netherlands: North-Holland / New York, N.Y.: Elsevier Science (1982).

IFIP Working Group 8.1会议论文集。描述了13个不同的方法，并且将其应用到一个标准的基准问题中。其中的一个方法就是NIAM（参见[13.28]）；这篇论文一定是最早讨论NIAM方法的文章之一。这本书还包括一些所提出的方法的回顾，其中也包括了NIAM模型。

13.30 M. P. Papazoglou: “Unraveling the Semantics of Conceptual Schemas,” *CACM* 38, No. 9 (September 1995).

这篇文章提出了一种称为元数据查询的方法——即根据数据库中数据的意思进行的查询，换句话说，就是根据概念模式本身进行的查询。这样的查询的一个例子是“什么是永久雇员？”。

13.31 Joan Peckham and Fred Maryanski: “Semantic Data Models,” *ACM Comp. Surv. 20*, No. 3 (September 1988).

另一个综述（参见文献[13.22]）。

13.32 Paul Reed: “The Unified Modeling Language Takes Shape,” *DBMS 11*, No. 8 (July 1998).

统一建模语言（Unified Modeling Language，UML），是另一个支持应用设计和开发的图符号（换句话说，在其中通过画图来开发应用程序）。它还可以用来开发SQL模式。

注意：UML很可能会在商业上很重要，部分原因是它被对象管理组 OMG采纳为标准，UML具有一种很强的对象风格。它已经被一组商业产品所支持。

UML支持数据和过程的建模（从这一点考虑，它超出了 E/R建模的范围），但是在完整性约束方面它似乎并没有说明什么（文献 [13.32]中标题为“ 从模型到代码：企业规则 ”一节中根本没有提到declarative这个术语！然而，它很重视利用过程应用代码来实现“ 处理 ”。以下是直接的引述：“ UML形式化地描述了为大家所熟知的对象概念：即现实世界的对象，最好作为既包含数据又包含函数的自含实体加以建模 ”。还有：“ 很明显，从一个历史发展的角度来看，数据和函数的分离导致了软件开发的脆弱 ”。这些评论从一个应用角度来讲可能是合理的，但是从一个数据库角度来看并不一定就合理。例如，参见文献[24.29]。

UML由Booch的“ Booch理论 ”[13.4]、Rumbaugh的OMT[13.3]和Jacobson的OOSE[13.23]的早期工作发展而来。Booch、Rumbaugh和Jacobson最近已经出版了一系列关于UML的书，这些书毫无疑问都是权威性的参考书，包括：《The Unified Modeling Language User Guide》、《The Unified Modeling Language Reference Manual》和《The Unified Software Development Process》所有的这三本书都是由 Addison-Wesley在1999年出版的。

13.33 H. A. Schmid and J. R. Swenson: “ On the Semantics of the Relational Data Base Model,” Proc. 1975 ACM SIGMOD Int. Conf. on Management of Data, San Jose, Calif. (May 1975).

这篇文章在Chen的E/R模型[13.5]之前提出了一个“ 基本的语义模型 ”，但是事实上它与E/R模型非常相似（当然，除了在术语的使用上； Schmid和Swenson分别用独立对象、依赖对象和关联替换了 Chen的常规实体、弱实体和联系的术语）。

13.34 J. F. Sowa: *Conceptual Structures: Information Processing in Mind and Machine*. Reading, Mass.: Addison-Wesley (1984).

这本书并不是专门讨论数据库系统的，它可以说是讨论知识表示和处理的一般问题。然而，这本书的一部分是直接与本章所讨论的内容相关的（下面的评论是基于 Sowa在1990年或1990年左右关于将“ 概念结构 ”应用于语义建模的生动介绍的）。E/R图和类似的表现形式中一个主要的问题是它们没有正式的逻辑严密性。因此，它们不能处理某些重要的设计特点 —— 特别是量词，这涉及到完整性约束的处理问题 —— 而形式逻辑就可以处理这些内容（量词是由 Frege在1879年提出的，这使得 E/R图只能成为“ 一种 1879年以前的逻辑 ”）。但是形式逻辑不易为接受；正如 Sowa所说：“ 谓词演算与知识表示语言非常相似 ”。概念图是一种可读的、严格的图符号，它可以表示整个逻辑。因此它们（ Sowa称）比E/R图和类似的图表更适合用于语义建模。

13.35 J. M. Smith and D. C. P. Smith: “ Database Abstractions: Aggregation,” *CACM 20*, No. 6 (June 1977).

参见下面的文献[13.36]

13.36 J. M. Smith and D. C. P. Smith: “ Database Abstractions: Aggregation and Generalization,” *ACM TODS 2*, No. 2 (June 1977).

这两篇论文[13.35]和[13.36]所提出的内容对 RM/T[13.6]有很重要的影响，特别是在实体子类型和超类型方面。

13.37 Veda C. Storey: “ Understanding Semantic Relationships,” *The VLDB Journal 2,* No. 4 (October 1993).

引自摘要：“语义数据模型应用抽象如 [子类型]、聚合以及关联的方式 [在数据库领域]得到了很好的发展。除了这些常见的联系，许多其它的语义联系也被研究者在其它领域规则中辨识出来，如语言学、逻辑和认知心理学领域。这篇文章探讨了一些这样的联系并且讨论了……它们对数据库设计的影响。”

13.38 B. Sundgren: “ The Infological Approach to Data Bases,” in J. W. Klimbie and K. L. Koffeman (eds.), *Data Base Management*. Amsterdam, Netherlands: North-Holland / New York, N.Y.: Elsevier Science (1974).

“信息逻辑方法（infological approach）”是最早被提出的逻辑建模方法之一。它在Scandinavia数据库设计领域成功地应用了许多年。

13.39 Dan Tasker: *Fourth Generation Data: A Guide to Data Analysis for New and Old Systems*. Sydney, Australia: Prentice-Hall of Australia Pty., Ltd. (1989).

数据库设计的实用指南，它将重点放在单个数据项（ data item）上（即放在域上）。数据项被分为三种基本种类：标签、数量和描述。标签项代表实体；在关系术语中，它与主码和外码相对应。数量项代表一个范围 (scale) (可能是一个时间列范围)的数量、量度或状态，并且用于常规的算术操作。描述项是所有剩下的部分（当然，对这种分类方案的论述很详细，而这里的简要介绍则不能完全概括）。这本书还详细介绍了每一种类型。这些讨论并不总是“纯关系”的——例如，Tasker对“域”这个词的使用就与其在关系上的使用不完全相同——但是这本书提供了很多可行的实际建议。

13.40 Toby J. Teorey and James P. Fry: *Design of Database Structures*. Englewood Cliffs, N.J.: Prentice-Hall (1982).

是一本有关数据库设计方方面面的教科书。这本书分为五部分：引言、概念设计、实现设计（即将概念设计映射到一个具体的 DBMS可以理解的结构上）、物理设计和特殊的设计问题。

13.41 Toby J. Teorey, Dongqing Yang, and James P. Fry: “A Logical Design Methodology for Relational Databases Using the Extended Entity -Relationship Model,” ACM Comp. Surv. 18, No. 2 (June 1986).

论文题目为“扩展的 E/R模型”提出了对实体类型层次、空值（ null）（参见第18章）和包含两个以上参与者的联系的支持。

13.42 Toby J. Teorey: *Database Modeling and Design: The Entity-Relationship Approach* (3rd edition). San Francisco, Calif.: Morgan Kaufmann (1998).

近期出版的一本关于数据库设计中 E/R应用和“扩展”的E/R概念[13.41]的教科书。