第7章
Database System Concepts, 6E

数据库设计和 E-R 模型

到现在为止，在本书中我们已经假想了一个给定的数据库模式并研究了如何表述查询和更新。我们现在考虑究竟如何设计一个数据库模式。在本章中，我们关注于实体 - 联系（E-R）数据模型，它提供了一个找出在数据库中表示的实体以及实体间如何关联的方法。最终，数据库设计将会表示为一个关系数据库设计和一个与之关联的约束集合。本章讲述一个 E-R 设计如何转换成一个关系模式的集合以及如何在该设计中找到某些约束。然后，第 8 章详细考查一个关系模式的集合是否为一个好的或不好的数据库设计，并研究使用更广的约束集合来构建好的设计的过程。这两章覆盖数据库设计的基本概念。

7.1 设计过程概览

构建一个数据库应用是一个复杂的任务，包括设计数据库模式，设计访问和更新数据的程序，以及设计控制数据访问的安全模式。用户的需求在设计过程中扮演一个中心角色。本章主要关注于数据库模式的设计，不过本章后面会简要地概括其他几个设计任务。

设计一个完整的数据库应用环境，并满足被建模企业的需求，需要关注广泛的问题。数据库预期用途的这些其他方面在物理级、逻辑级和视图级影响着各种各样的设计选择。

7.1.1 设计阶段

259 对于小型的应用，由一个理解应用需求的数据库设计者直接决定要构建的关系、关系的属性以及其上的约束，这样也许是可行的。但是，这种直接的设计过程在现实的应用中是困难的，由于现实的应用常常很复杂，通常没有一个人能够理解应用的所有数据需求。数据库设计者必须与应用的用户进行交互以理解应用的需求，把它们以用户能够理解的高级别的形式表示出来，然后再将需求转化为较低级别的设计。一个高层数据模型为数据库设计者提供了一个概念框架，在该框架中以系统的方式定义了数据库用户的数据需求，以及满足这些需求的数据库结构。

- 数据库设计的最初阶段需要完整地刻画未来数据库用户的数据需求。为完成这个任务，数据库设计者需要同应用领域的专家和用户进行深入的沟通。这一阶段的产品是用户需求规格说明。虽然存在图形方式的用户需求表示技术，但是在本章中，我们仅限于采用文字的方式来描述用户需求。
- 接下来，设计者选择数据模型，并采用所选数据模型的概念将这些需求转化为数据库的概念模式。在此**概念设计**（conceptual-design）阶段所产生的模式提供了一个对企业的详细综述。我们在本章中将研究的实体 - 联系模型通常用于表示概念设计。用实体 - 联系模型的术语来说，概念模式定义了数据库中表示的实体、实体的属性、实体之间的联系，以及实体和联系上的约束。通常，概念设计阶段会导致实体 - 联系图的构建，它提供了对模式的图形化描述。

  设计者检查此模式以确保所有数据需求都满足，并且互相不冲突。她还可以检查该设计以去除冗余的特征。在这个阶段，她关注的是描述数据及其联系，而不是定义物理存储细节。
- 完善的概念模式还指明企业的功能需求。在**功能需求规格说明**（specification of functional requirement）中，用户描述将在数据上进行的各类操作（或事务）。操作的例子包括修改或更

新数据，搜索并取回特定数据，以及删除数据。在概念设计的这一阶段，设计者可以检查所设计的模式，以确保其满足功能需求。

- 从抽象数据模型到数据库实现的转换过程在最后两个设计阶段中进行。
  - □ 在**逻辑设计阶段**（logical-design phase）中，设计者将高层概念模式映射到将使用的数据库系统的实现数据模型上。实现数据模型通常是关系数据模型，该阶段通常包括将以实体 - 联系模型定义的概念模式映射到关系模式。 260
  - □ 最后，设计者将所得到的系统特定的数据库模式使用到后续的**物理设计阶段**（physical-design phase）中。在该阶段，指明数据库的物理特征，这些特征包括文件组织格式和索引结构的选择，我们将在第 10 章和第 11 章讨论这些内容。

在应用建立之后，要改变数据库的物理模式相对地比较简单。但是，由于可能影响到应用程序代码中散布的大量的查询和更新操作，因此改变逻辑模式的任务执行起来常常更加困难。因此，在建立后续的数据库应用之前，慎重实施数据库设计阶段是非常重要的。

7.1.2　设计选择

数据库设计过程的一个主要部分是决定如何在设计中表示各种类型的“事物”，比如人、地方、产品，诸如此类。我们使用实体（entity）这个术语来指示所有可明确识别的个体。在一个大学数据库中，实体的例子将包括教师、学生、系、课程和开课[⊖]。这些各种各样的实体以多种方式互相关联，而所有这些方式都需要在数据库设计中反映出来。例如，一名学生在一次开课中选课，而一名教师在一次开课中授课，授课和选课就是实体间联系的实例。

在设计一个数据库模式的时候，我们必须确保避免两个主要的缺陷。

- **冗余**：一个不好的设计可能会重复信息。例如，如果对于每一次开课我们都存储课程编号和课程名称，那么对于每一次开课我们就冗余地（即多次地、不必要地）存储了课程名称。对每次开课仅存储课程编号，并在一个课程实体中将课程名称和课程编号关联一次就足够了。

  冗余也可能出现在关系模式中。在目前我们所使用的大学的例子中，我们有一个开课信息的关系和一个课程信息的关系。假设换一个做法，我们只有一个关系，对一门课程的每一次开课我们将全部课程信息（课程编号、课程名、系名、学分）重复一次。很明显，关于课程的信息将冗余地存储。

  信息的这种冗余表达的最大问题是当对一条信息进行更新但没有将这条信息的所有拷贝都更新时，这条信息的拷贝会变得不一致。例如，拥有同一个课程编号的几次不同的开课可能会拥有不同的课程名称，于是会搞不清楚课程的正确名称是什么。理想的情况下，信息应该只出现在一个地方。 261
- **不完整**：一个不好的设计可能会使得企事业机构的某些方面难于甚至无法建模。例如，假设在上述案例（1）中，我们只有对应于开课的实体，而没有对应于课程的实体。从关系的角度说，这就是假设我们有单个关系，对一门课程的每一次开课都重复存储课程的所有信息。那么一门新课程的信息将无法表示，除非开设了该课程。我们可能会尝试将开课信息设置为空值的方法来解决这个有问题的设计。这种绕开问题的措施不仅不吸引人，而且有可能由于主码约束而无法实行。

仅仅避免不好的设计是不够的。可能存在大量好的设计，我们必须从中选择一个。作为一个简单的例子，考虑买了某产品的一个客户。该产品的销售是客户和产品之间的联系吗？还是销售本身是一个与客户和产品都相关的实体？这个选择，虽然简单，却可能对企业某些方面建模的好坏有很大的影响。考虑为一个现实企业中的大量实体和联系做类似这样的选择的需要，不难看出数据库设计是一个很有挑战性的问题。事实上我们将看到，它需要科学和“好的品味”的结合。

⊖　一门课程可能在多个学期开设，同样也可能一个学期中开设多次，我们称某课程的每次开设为一个课程段。

7.2 实体－联系模型

实体－联系（entity-relationship，E-R）数据模型的提出旨在方便数据库的设计，它是通过允许定义代表数据库全局逻辑结构的企业模式实现的。

E-R 模型在将现实世界企业的含义和交互映射到概念模式上非常有用，因此，许多数据库设计工具都利用了 E-R 模型的概念。E-R 数据模型采用了三个基本概念：实体集、联系集和属性，我们首先对此进行学习；E-R 模型还有一个相关联的图形表示（E-R 图），我们在本章后面学习。

7.2.1 实体集

实体（entity）是现实世界中可区别于所有其他对象的一个“事物”或“对象”。例如，大学中的每个人都是一个实体。每个实体有一组性质，其中一些性质的值可以唯一地标识一个实体。例如，一
262 个人可能具有 *person_id* 性质唯一标识这个人。因此，*person_id* 的值 677－89－9011 将唯一标识大学中一个特定的人。与此类似，课程也可以看作实体，而 *course_id* 唯一标识了大学中的某个课程实体。实体可以是实实在在的，如人或书；也可以是抽象的，如课程、课程段开课或者机票预订。

实体集（entity set）是相同类型即具有相同性质（或属性）的一个实体集合。例如，一所给定大学的所有教师的集合可定义为实体集 *instructor*。类似地，实体集 *student* 可以表示大学中所有学生的集合。

在建模的过程中，我么通常抽象地使用术语实体集，而不是指某个个别实体的特别集合。我们用术语实体集的**外延**（extension）来指属于实体集的实体的实际集合。因此，大学中实际教师的集合构成了实体集 *instructor* 的外延。我们在第 2 章看到过的联系和联系实例之间的区别和上述区别类似。

实体集不必互不相交。例如，可以定义大学里所有人的实体集（*person*）。一个 *person* 实体可以是 *instructor* 实体，可以是 *student* 实体，可以既是 *instructor* 实体又是 *student* 实体，也可以都不是。

实体通过一组**属性**（attribute）来表示。属性是实体集中每个成员所拥有的描述性性质。为某实体集指定一个属性表明数据库为该实体集中每个实体存储相似的信息；但每个实体在每个属性上都有各自的值。实体集 *instructor* 可能具有属性 *ID*、*name*、*dept_name*、和 *salary*。在现实生活中，可能还有更多的属性，如街道号、房间号、州、邮政编码和国家，但是为了使我们的例子简单，我们省略了这些属性。*course* 实体集可能的属性有 *course_id*、*title*、*dept_name* 和 *credits*。

每个实体的每个属性都有一个值（value）。例如，一个特定的 *instructor* 实体可能 *ID* 的值为 12121，*name* 的值为吴，*dept_name* 的值为金融，*salary* 的值为 90 000。

ID 属性用来唯一地标识教师，因为可能会有多个教师拥有相同的名字。在美国，许多企业发现将一个人的社会保障号[⊖]用作其值唯一标识该人的属性很方便。一般来说，大学必须给每个教师创建和分配一个唯一的标识符。

因此，数据库包括一组实体集，每个实体集包括任意数量的相同类型的实体。图 7-1 为一个大学数据库的一部分，其中有两个实体集：*instructor* 和 *student*。为了使图示简单，只显示了两个实体集的一部分属性。

一个大学数据库可能包含许多其他的实体集。例如，除了跟踪记录教师和学生外，大学还具有课
263 程信息，用实体集 *course* 来表示，它包括属性 *account_number*、*course_id*、*title*、*dept_name* 和 *credits*。在真实环境中，一个大学数据库可能会包含数十个实体集。

7.2.2 联系集

联系（relationship）是指多个实体间的相互关联。例如，我们可以定义关联教师 Katz 和学生 Shankar 的联系 *advisor*。这一联系指明 Katz 是学生 Shankar 的导师。

⊖ 在美国，政府分配给国家中的每一个人一个唯一的号码，称为社会保障号，用来唯一地标识一个人。任何一个人都仅有一个这样的社会保障号，并且没有两个人会拥有相同的社会保障号。

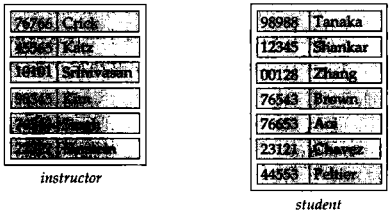


图 7-1　实体集 *instructor* 和 *student*

联系集(relationship set)是相同类型联系的集合。正规地说，联系集是 $n \geq 2$ 个(可能相同的)实体集上的数学关系。如果 E_1，E_2，…，E_n 为实体集，那么联系集 R 是

$$\{(e_1, e_2, \cdots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \cdots, e_n \in E_n\}$$

的一个子集，而$(e_1, e_2, \cdots, e_n)$是一个联系。

考虑图 7-1 中的两个实体集 *instructor* 和 *student*。我们定义联系集 *advisor* 来表示教师及学生之间的关联。这一关联如图 7-2 所示。

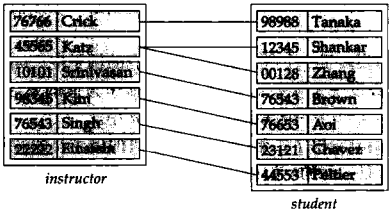


图 7-2　联系集 *advisor*

再看另一个例子，考虑实体集 *student* 和 *section*。我们可以定义联系集 *takes* 来表示学生和该学生所注册的开课之间的关联。

实体集之间的关联称为参与；也就是说，实体集 E_1，E_2，…，E_n **参与**(participate)联系集 R。E-R 模式中的一个**联系实例**(relationship instance)表示在所建模的现实世界企业中命名实体间的一个关联。例如，一个教师 *ID* 为 45565 的 *instructor* 实体 Katz 和一个学生 *ID* 为 12345 的 *student* 实体 Shankar 参与到 *advisor* 的一个联系实例中。这一联系实例表示在大学中教师 Katz 指导学生 Shankar。

实体在联系中扮演的功能称为实体的**角色**(role)。由于参与一个联系集的实体集通常是互异的，因此角色是隐含的并且一般并不指定。但是，当联系的含义需要解释时角色是很有用的。当参与联系 264
集的实体集并非互异的时候就是这种情况；也就是说，同样的实体集以不同的角色参与一个联系集多于一次。在这类联系集中，即有时称作**自环的**(recursive)联系集中，有必要用显式的角色名来指明实体是如何参与联系实例的。例如，考虑记录大学开设的所有课程的信息的实体集 *course*。我们用 *course* 实体的有序对来建模联系集 *prereq*，以描述一门课程($C2$)是另一门课程($C1$)的**先修课**。每对课程中的第一门课程具有课程 $C1$ 的角色，而第二门课程具有先修课 $C2$ 的角色。按照这种方式，所有的 *prereq* 联系通过($C1$，$C2$)对来表示，排除了($C2$，$C1$)对。

联系也可以具有描述性属性(descriptive attribute)。考虑实体集 *instructor* 和 *student* 之间的联系集 *advisor*。我们可以将属性 *date* 与该联系关联起来，以表示教师成为学生的导师的日期。教师 Katz 对应的实体和学生 Shankar 对应的实体之间的联系 *advisor* 的属性 *date* 的值为“10 June 2007”，表示 Katz 于 2007 年 6 月 10 日成为 Shankar 的导师。

图 7-3 所示为具有描述性属性 *date* 的联系集 *advisor*。注意，Katz 在两个不同的日期成为了两名学生的导师。

作为联系的描述性属性的一个更实际的例子，考虑实体集 *student* 和 *section* 参与一个联系集 *takes*。我们也许希望在这个联系中用一个描述性属性 *grade*，来记录学生在这门课中取得的成绩。我们同样可以用一个描述性的属性 *for_credit* 来记录学生在这门课中是选修还是旁听(或出席)的情况。

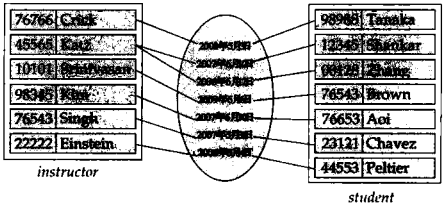


图 7-3 *date* 作为联系集 *advisor* 的属性

给定的联系集中的一个联系实例必须是由其参与实体唯一标识的，而不必使用描述属性。为了理
265 解这一点，假设我们要对一个教师成为一个特定学生的导师的所有日期建模。单值的属性 *date* 只能保存一个日期。我们不能通过同一个教师和学生之间的多个联系实例来表示多个日期，因为这些联系实例仅使用参与的实体是无法唯一标识的。正确的处理方法是创建一个多值属性 *date*，它可以保存所有的日期。

相同的实体集可能会参与到多于一个联系集中。在我们的例子中，*instructor* 和 *student* 实体集参与到联系集 *advisor* 中。另外，假设每个学生必须有一名教师作为他的系导师(本科或研究生)，那么 *instructor* 和 *student* 实体集将参与到另一个联系集 *dept_advisor* 中。

联系集 *advisor* 和 *dept_advisor* 给出了**二元**(binary)联系集的例子，即涉及两个实体集的联系集。数据库系统中的大部分联系集都是二元的。然而，有时联系集会涉及多于两个实体集。

例如，假设我们有一个代表在大学内开展的所有研究项目的实体集 *project*，考虑实体集 *instrustor*、*student* 和 *project*。每个项目可以有多个参与的学生和多个参与的教师。另外，每个参与项目的学生必须有一个教师指导他在项目中的工作。目前，我们忽略项目和教师以及项目和学生这两个关联，而关注哪个教师在一个特定项目上指导哪个学生。为了表达这个信息，我们通过关联 *proj_guide* 将三个实体集联系到一起，它表示某个学生在某个项目上接收了某个教师的指导。

注意，一个学生可以在不同的项目中有不同的教师作为导师，不能将这个联系描述成学生与教师
266 之间的二元关系。

参与联系集的实体集的数目称为联系集的**度**(degree)。二元联系集的度为2；三元联系集的度为3。

7.2.3 属性

每个属性都有一个可取值的集合，称为该属性的**域**(domain)，或者**值集**(value set)。*course_id* 属性的域可能是特定长度的所有文本字符串的集合。类似地，属性 *semester* 的域可能是集合{秋，冬，春，夏}中的字符串。

正规地说，实体集的属性是将实体集映射到域的函数。由于一个实体集可能有多个属性，因此每个实体可以用一组(属性，数据值)对来表示，实体集的每个属性对应一个这样的对。例如，某个 *instructor* 实体可以用集合{(*ID*，76766)，(*name*，Crick)，(*dept_name*，生物)，(*salary*，72000)}来描述，该实体描述了一个叫 Crick 的人，他的教师编号为 76766，是生物系的成员，工资为 $72 000。从这里我们可以看出抽象模式与被建模的实际企业的结合。用来描述实体的属性值构成存储在数据库中的数据的一个重要部分。

E-R 模型中的属性可以按照如下的属性类型来进行划分：

- **简单**(simple)和**复合**(composite)属性。在我们的例子中，迄今为止出现的属性都是简单属性，也就是说，它们不能划分为更小的部分。另一方面，**复合**(composite)属性可以再划分为更小的部分(即其他属性)。例如，属性 *name* 可设计为一个包括 *first_name*、*middle_initial* 和 *last_name*

的复合属性。如果一个用户希望在一些场景中引用完整的属性，而在另外的场景中仅引用属性的一部分，则在设计模式中使用复合属性是一个好的选择。假设我们要给 *student* 实体集增加一个地址。地址可定义为包含属性 *street*、*city*、*state* 和 *zip_code* ⊖的复合属性 *address*。复合属性帮助我们把相关属性聚集起来，使模型更清晰。

注意，复合属性可以是有层次的。在复合属性 *address* 中，其子属性 *street* 可以进一步分为 *street_number*、*street_name* 和 *apartment_number*。图 7-4 描述了 *instructor* 实体集的这些复合属性的例子。

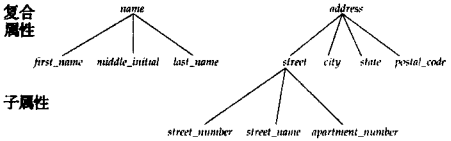


图 7-4　复合属性教师 *name* 和 *address*

- **单值**(single-valued)**和多值**(multivalued)**属性**。我们的例子中的属性对一个特定实体都只有单独的一个值。例如，对某个特定的学生实体而言，*student_ID* 属性只对应于一个学生 *ID*。这样的属性称作是**单值**(single valued)的。而在某些情况下对某个特定实体而言，一个属性可能对应于一组值。假设我们往 *instructor* 实体集添加一个 *phone_number* 属性，每个教师可以有零个、一 267
个或多个电话号码，不同的教师可以有不同数量的电话。这样的属性称作是**多值**(multivalued)的。作为另一个例子，我们可以往实体集 *instructor* 中添加一个属性 *dependent_name*，它列出所有的眷属。这个属性将是多值的，因为任何一个特定的教师可能有零个、一个或多个眷属。

  为了表示一个属性是多值的，我们用花括号将属性名括住，例如：{*phone_number*} 或者 {*dependent_name*}。

  在适当的情况下，可以对一个多值属性的取值数目设置上、下界。例如，一所大学可能将单个教师的电话号码个数限制在两个以内。在这个例子中设置限制表明 *instructor* 实体集的 *phone_number* 属性可以有 0 ~ 2 个值。

- **派生**(derived)**属性**。这类属性的值可以从别的相关属性或实体派生出来。例如，让我们假设 *instructor* 实体集有一个属性 *students_advised*，表示一个教师指导了多少个学生。我们可以通过统计与一个教师相关联的所有 *student* 实体的数目来得到这个属性的值。

  又如，假设 *instructor* 实体集具有属性 *age*，表示教师的年龄。如果 *instructor* 实体集还具有属性 *date_of_birth*，我们就可以从当前的日期和 *date_of_birth* 计算出 *age*。因此 *age* 就是派生属性。在这里，*date_of_birth* 可以称为基属性，或存储的属性。派生属性的值不存储，而是在需要时计算出来。

当实体在某个属性上没有值时使用**空**(null)值。空值可以表示“不适用”，即该实体的这个属性不存在值。例如，一个人可能没有中间名字。空还可以用来表示属性值未知。未知的值可能是*缺失的*(值存在，但我们没有该信息)，或*不知道的*(我们并不知道该值是否确实存在)。

例如，如果一个特定的教师的 *name* 值是空，我们推测这个值是缺失的，因为每个教师肯定有一个名字。*apartment_number* 属性的空值可能意味着地址不包括房间号(不适用)，或房间号是存在的但是我 268
们不知道是什么(缺失的)，或者我们不知道房间号是否是该教师的地址的一部分(不知道的)。

7.3　约束

E-R 企业模式可以定义一些数据库中的数据必须要满足的约束。这一节讨论映射基数以及参与约束。

⊖ 我们假定使用美国的地址格式，其中包括一个称作 zip code 的数字邮政编码。

7.3.1 映射基数

映射基数(mapping cardinality)，或基数比率，表示一个实体通过一个联系集能关联的实体的个数。

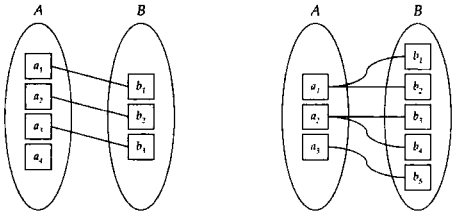
映射基数在描述二元联系集时非常有用，尽管它们可以用于描述涉及多于两个实体集的联系集。在这一节中，我们将只集中在二元联系集上。

对于实体集 A 和 B 之间的二元联系集 R 来说，映射基数必然是以下情况之一：

- **一对一**(one-to-one)。A 中的一个实体至多与 B 中的一个实体相关联，并且 B 中的一个实体也至多与 A 中的一个实体相关联(如图 7-5a 所示)。
- **一对多**(one-to-many)。A 中的一个实体可以与 B 中的任意数目(零个或多个)实体相关联，而 B 中的一个实体至多与 A 中的一个实体相关联(如图 7-5b 所示)。
- **多对一**(many-to-one)。A 中的一个实体至多与 B 中的一个实体相关联，而 B 中的一个实体可以与 A 中任意数目(零个或多个)实体相关联(如图 7-6a 所示)。
- **多对多**(many-to-many)。A 中的一个实体可以与 B 中任意数目(零个或多个)实体相关联，而且 B 中的一个实体也可以与 A 中任意数目(零个或多个)实体相关联(如图 7-6b 所示)。

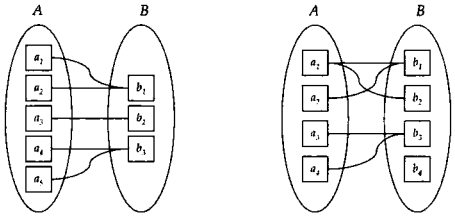
显然，一个特定联系集的适当的映射基数依赖于该联系集所建模的现实世界的情况。

作为例子，考虑 *advisor* 联系集。如果在一所特定的大学中，一名学生只能由一名教师指导，而一名教师可以指导多个学生，那么 *instructor* 到 *student* 的联系集是一对多的。如果一名学生可以由多名教师指导(比如学生可以由多名教师共同指导)，那么此联系集是多对多的。



a) 一对一　　b) 一对多

图 7-5　映射基数



a) 多对一　　b) 多对多

图 7-6　映射基数

7.3.2 参与约束

如果实体集 E 中的每个实体都参与到联系集 R 的至少一个联系中，实体集 E 在联系集 R 中的参与称为**全部**(total)的。如果 E 中只有部分实体参与到 R 的联系中，实体集 E 到联系集 R 的参与称为**部分**(partial)的。在图 7-5a 中，B 在联系集中的参与是全部的，而 A 在联系集中的参与是部分的；在图 7-5b 中，A 和 B 在联系集中的参与都是全部的。

例如，我们期望每个 *student* 实体通过 *advisor* 联系同至少一名教师相联系，因而 *student* 在联系集 *advisor* 中的参与是全部的。相反地，一个 *instructor* 不是必须要指导一个学生。因此，很可能只有一部分 *instructor* 实体通过 *advisor* 联系同 *student* 实体集相关联，于是 *instructor* 在 *advisor* 联系集中的参与是部分的。 269 | 270

7.3.3　码

我们必须有一个区分给定实体集中的实体的方法。从概念上来说，各个实体是互异的；但从数据库的观点来看，它们的区别必须通过其属性来表明。

因此，一个实体的属性的值必须可以唯一标识该实体。也就是说，在一个实体集中不允许两个实体对于所有属性都具有完全相同的值。

2.3 节定义的关系模式的码的概念直接适用于实体集。即实体的码是一个足以区分每个实体的属性集。关系模式中的超码、候选码、主码的概念同样适用于实体集。

码同样用于唯一地标识联系，并从而将联系互相区分开来。我们在下面定义联系的码的相应概念。

实体集的主码使得我们可以区分实体集中不同的实体。我们需要一种类似的机制来区分联系集中不同的联系。

设 R 是一个涉及实体集 E_1，E_2，…，E_n 的联系集。设主码 (E_i) 代表构成实体集 E_i 的主码的属性集合。目前我们假设所有主码的属性名是互不相同的。联系集主码的构成依赖于同联系集 R 相关联的属性集合。

如果联系集 R 没有属性与之相关联，那么属性集合

$$primary\text{-}key(E_1) \cup primary\text{-}key(E_2) \cup \cdots \cup primary\text{-}key(E_n)$$

描述了集合 R 中的一个联系。

如果联系集 R 有属性 a_1，a_2，…，a_m 与之相关联，那么属性集合

$$primary\text{-}key(E_1) \cup primary\text{-}key(E_2) \cup \cdots \cup primary\text{-}key(E_n) \cup \{a_1, a_2, \cdots, a_m\}$$

描述了集合 R 中的一个联系。

在以上两种情况下，属性集合

$$primary\text{-}key(E_1) \cup primary\text{-}key(E_2) \cup \cdots \cup primary\text{-}key(E_n)$$

构成了联系集的一个超码。

如果实体集间主码的属性名称不是互不相同的，重命名这些属性以区分它们；实体集的名字加上属性名可以构成唯一的名称。如果一个实体集不止一次参与某个联系集（如 7.2.2 节中的 *prereq* 联系），则使用角色名代替实体集名构成唯一的属性名。 271

联系集的主码结构依赖于联系集的映射基数。例如，考虑在 7.2.2 节中的实体集 *instructor* 和 *student* 以及具有属性 *date* 的联系集 *advisor*。假设联系集是多对多的，那么 *advisor* 的主码由 *instructor* 和 *student* 的主码的并集组成。如果联系是从 *student* 到 *instructor* 多对一的，即每个学生最多只能有一个导师，则 *student* 的主码就是 *advisor* 的主码。而如果一名教师只能指导一名学生，即联系是从 *instructor* 到 *student* 多对一的，则 *instructor* 的主码就是 *advisor* 的主码。对于一对一的联系，两个候选码中的任意一个可以用作主码。

对于非二元联系，如果没有基数的限制，那么在本节开始时描述的超码就是唯一的候选码，并被选为主码。如果有基数的限制，主码的选择就复杂多了。因为我们还没有讨论如何在非二元关系中描述基数约束，所以我们在本章中不对此做更多的讨论。我们将在 7.5.5 节和 8.4 节详细地考虑这个问题。

7.4　从实体集中删除冗余属性

当我们使用 E-R 模型设计数据库时，我们通常从确定那些应当包含的实体集开始。例如，在我们迄今所讨论的大学机构中，我们想要包含如 *student* 和 *instructor* 等实体集。当决定好实体集后，我们必须挑选适当的属性，这些属性要表示我们在数据库中所捕获的不同的值。在大学机构中，我们为 instructor 实体集设计了包括 *ID*、*name*、*dept_name* 以及 *salary* 几个属性，我们还可以增加 *phone_number*、*office_number*、*home_page* 等属性。要包含哪些属性的选择决定于了解企业结构的设计者。

一旦选择好实体和它们相应的属性，不同实体间的联系集就建立起来了。这些联系集有可能会导

致不同实体集中的属性冗余，并需要将其从原始实体集中删除。为了说明这一点，考虑实体集 *instructor* 和 *department*：

- 实体集 *instructor* 包含属性 *ID*、*name*、*dept_name* 以及 *salary*，其中 *ID* 构成主码。
- 实体集 *department* 包含属性 *dept_name*、*building* 以及 *budget*，其中 *dept_name* 构成主码。

272 我们用关联 *instructor* 和 *department* 的联系集 *inst_dept* 对每个教师都有一个关联的系的情况建模。

属性 *dept_name* 在两个实体集中都出现了。由于它是实体集 *department* 的主码，因此它在实体集 *instructor* 中是冗余的，需要将其移除。

从实体集 *instructor* 中移除属性 *dept_name* 可能不是那么直观，因为我们在前几章所用到的关系 *instructor* 中具有 *dept_name* 属性。我们将在后面看到，当我们从 E-R 图构建一个关系模式时，只有当每个教师最多只与一个系关联时，属性 *dept_name* 才会添加到关系 *instructor* 中。如果一个教师有多个关联的系时，教师与系之间的联系会记录在一个单独的关系 *inst_dept* 中。

将教师和系之间的关联统一看成联系，而不是 *instructor* 的一个属性，使得逻辑关系明确，并有助于避免过早地假设每个教师只与一个系关联。

类似地，实体集 *student* 通过联系集 *student_dept* 与实体集 *department* 关联，因而 *student* 中不需要 *dept_name* 属性。

作为另一个例子，考虑开课(section)和开课的时段。每个时段都由 *time_slot_id* 标识，并且和上课时间的集合相关联，每次上课时间都由星期几、开始时间以及结束时间标识。我们打算使用多值复合属性对上课时间集合建模。假设我们对实体集 *section* 和 *time_slot* 按以下方式建模：

- 实体集 *section* 包含属性 *course_id*、*sec_id*、*semester*、*year*、*building*、*room_number* 以及 *time_slot_id*，其中(*course_id*、*sec_id*、*year*、*semester*)构成主码。
- 实体集 time_slot 包含主码 ⊖属性 *time_slot_id*，以及一个多值复合属性 {(*day*, *start_time*, *end_time*)} ⊜。

这些实体通过联系集 *sec_time_slot* 相互关联。

属性 *time_slot_id* 在两个实体集中均出现。由于它是实体集 *time_slot* 的主码，因此它在实体集 *section* 中是冗余的，并且需要将其删除。

作为最后的例子，假设我们有一个实体集 *classroom*，包含属性 *building*、*room_number* 以及 *capacity*，主码由 *building* 和 *room_number* 组成。再假设我们有一个联系集 *sec_class*，将 *section* 和 *classroom* 关联在一起。那么属性 {*building*, *room_number*} 在实体集 *section* 中是冗余的。

一个好的实体 - 联系设计不包含冗余的属性。对于我们的大学的例子，我们在下面列出实体集以

273 及它们的属性，主码以下划线标明。

- **classroom** ：包含属性(*building*、*room_number*、*capacity*)。
- **department** ：包含属性(*dept_name*、*building*、*budget*)。
- **course** ：包含属性(*course_id*、*title*、*credits*)。
- **instructor** ：包含属性(*ID*、*name*、*salary*)。
- **section** ：包含属性(*course_id*、*sec_id*、*semester*、*year*)。
- **student** ：包含属性(*ID*、*name*、*tot_cred*)。
- **time_slot** ：包含属性 (*time_slot_id*、{(*day*, *start_time*, *end_time*)})。

我们设计的联系集如下。

- **inst_dept**：关联教师和系。
- **stud_dept**：关联学生和系。
- **teaches**：关联教师和开课。

⊖ 我们将在后面看到由实体集 *time_slot* 构建的关系的主码包含 *day* 以及 *start_time*，然而，*day* 和 *start_time* 并不是实体集 *time_slot* 的主码一部分。

⊜ 我们可以给包含 *day*、*start_time* 以及 *end_time* 的复合属性选择一个名字，例如 *meeting*。

- **takes**：关联学生和开课，包含描述性属性 *grade*。
- **course_dept**：关联课程和系。
- **sec_course**：关联开课和课程。
- **sec_class**：关联开课和教室。
- **sec_time_slot**：关联开课和时段。
- **advisor** ：关联学生和教师。
- **prereq**：关联课程和先修课程。

你可以验证没有任何一个实体集包含由联系集而造成冗余的属性；另外，你可以验证我们此前在第 2 章的图 2-8 中看到的大学数据库关系模式中的所有的信息(除了约束)全部包含在上述的设计中，只是关系设计中的几个属性被 E-R 设计中的联系所替代。

7.5　实体 - 联系图

1.3.3 节曾经简要地介绍过，**E-R 图**(E-R diagram)可以图形化表示数据库的全局逻辑结构。E-R 图既简单又清晰，这些是致使 E-R 模型广泛使用的重要性质。

7.5.1　基本结构

E-R 图包括如下几个主要构件：　274

- **分成两部分的矩形**代表实体集。本书中有阴影的第一部分包含实体集的名字，第二部分包含实体集中所有属性的名字。
- **菱形**代表联系集。
- **未分割的矩形**代表联系集的属性。构成主码的属性以下划线标明。
- **线段**将实体集连接到联系集。
- **虚线**将联系集属性连接到联系集。
- **双线**显示实体在联系集中的参与度。
- **双菱形**代表连接到弱实体集的标志性联系集(我们将在 7.5.6 节讲述标志性联系集和弱实体集)。

考虑图 7-7 中的 E-R 图，它由通过二元联系集 *advisor* 关联的两个实体集 *instructor* 和 *student* 组成。同 *instructor* 相关联的属性为 *ID*、*name* 和 *salary*。同 *student* 相关联的属性为 *ID*、*name* 和 *tot_cred*。如图 7-7所示，实体集的属性中那些组成主码的属性以下划线标明。

如果一个联系集有关联的属性，那么我们将这些属性放入一个矩形中，并且用虚线将该矩形与代表联系集的菱形连接起来。例如，在图 7-8 中，我们有描述性属性 *date* 附带到联系集 *advisor* 上，表示教师成为导师的日期。

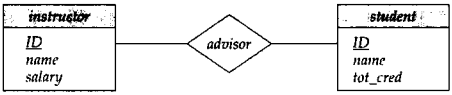


图 7-7　对应于教师和学生的 E-R 图

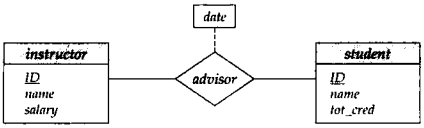


图 7-8　联系集上附带了一个属性的 E-R 图　275

7.5.2 映射基数

实体集 *instructor* 和 *student* 之间的联系集 *advisor* 可以是一对一、一对多、多对一或多对多的。为了区别这些类型，我们在所讨论的联系集和实体集之间画一个箭头(→)或一条线段(—)，如下所示。

- **一对一**：我们从联系集 *advisor* 向实体集 *instructor* 和 *student* 各画一个箭头(见图 7-9a)。这表示一名教师可以指导至多一名学生，并且一名学生可以有至多一位导师。
- **一对多**：我们从联系集 *advisor* 画一个箭头到实体集 *instructor*，以及一条线段到实体集 *student*(见图 7-9b)。这表示一名教师可以指导多名学生，但一名学生可以有至多一位导师。
- **多对一**：我们从联系集 *advisor* 画一条线段到实体集 *instructor*，以及一个箭头到实体集 *student*。这表示一名教师可以指导至多一名学生，但一名学生可以有多位导师。
- **多对多**：我们从联系集 *advisor* 向实体集 *instructor* 和 *student* 各画一条线段(见图 7-9c)。这表示一名教师可以指导多名学生，并且一名学生可以有多位导师。

276

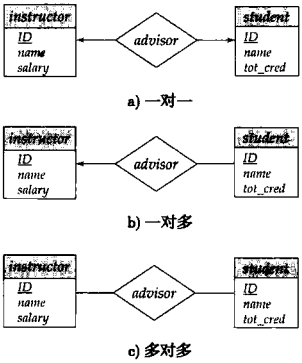


图 7-9 联系

E-R 图还提供了一种描述每个实体参与联系集中的联系的次数的更复杂的约束的方法。实体集和二元联系集之间的一条边可以有一个关联的最大和最小的映射基数，用 $l..h$ 的形式表示，其中 l 表示最小的映射基数，而 h 表示最大的映射基数。最小值为 1 表示这个实体集在该联系集中全部参与，即实体集中的每个实体在联系集中的至少一个联系中出现。最大值为 1 表示这个实体参与至多一个联系，而最大值为 * 代表没有限制。

例如，考虑图 7-10，在 *advisor* 和 *student* 之间的边有 1..1 的基数约束，意味着基数的最小值和最大值都是 1。也就是，每个学生必须有且仅有一个导师。从 *advisor* 到 *instructor* 边上的约束 0..* 说明教师可以有零个或多个学生。因此，*advisor* 联系是从 *instructor* 到 *student* 的一对多联系，更进一步地讲，*student* 在 *advisor* 联系中的参与是全部的，表示一个学生必须有一个导师。

图 7-10 联系集上的基数约束

很容易将左侧边上的 0..* 曲解为联系 *advisor* 是从 *instructor* 到 *student* 多对一的，而这正好和正确的解释相反。

如果两条边都有最大值 1，那么这个联系是一对一的。如果我们在左侧边上标明基数约束 1..*，我们就可以说每名教师必须指导至少一名学生。

图 7-10 中的 E-R 图的另一种画法是在基数约束的位置画一条从 *student* 到 *advisor* 的双线，以及一个从 *advisor* 到 *instructor* 的箭头。这种画法可以强制实施同图 7-10 中所示约束完全一样的约束。

7.5.3 复杂的属性

图 7-11 说明了怎样在 E-R 图中表示复合属性。这里一个具有子属性 *first_name*、*middle_initial* 和 *last_name* 的复合属性 *name* 代替了 *instructor* 的简单属性 *name*。再例如，假定我们给实体集 *instructor* 增加一个地址。地址可以定义为具有属性 *street*、*city*、*state* 和 *zip_code* 的

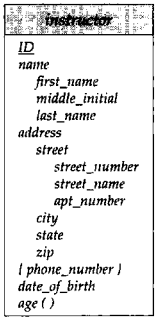


图 7-11 包含复合、多值和派生属性的 E-R 图

复合属性 *address*。属性 *street* 本身也是一个复合属性，其子属性为 *street_number*、*street_name* 和 *apartment_number*。

在图 7-11 还给出了一个由“{*phone_number*}”表示的多值属性 *phone_number* 和一个由“{*age*( )}”表示的派生属性 *age*。

7.5.4　角色

在 E-R 图中，我们通过在菱形和矩形之间的连线上进行标注来表示角色。图 7-12 给出了 *course* 实体集和 *prereq* 联系集之间的角色标识 *course_id* 和 *prereq_id*。

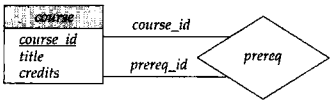


图 7-12　包含角色标识的 E-R 图

7.5.5　非二元的联系集

非二元的联系集也可以在 E-R 图中简单地表示。图 7-13 包含三个实体集 *instructor*、*student* 和 *project*，它们通过联系集 *proj_guide* 相关联。 277 ~ 278

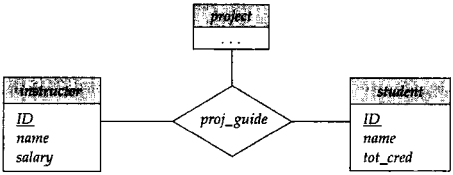


图 7-13　包含三元联系的 E-R 图

在非二元的联系集中，我们可以表示某些类型的多对一联系。假设一个 *student* 在每个项目上最多只能有一位导师。这种约束可用从 *proj_guide* 的边指向 *instructor* 的箭头来表示。

在一个联系集外我们至多允许一个箭头，因为在一个非二元的联系集外包含两个或更多箭头的 E-R 图可以用两种方法解释。假设实体集 A_1，A_2，…，A_n 之间有联系集 R，并且只有指向实体集 A_{i+1}，A_{i+2}，…，A_n 的边是箭头。那么，两种可能的解释为：

- 来自 A_1，A_2，…，A_i 的实体的一个特定组合可以和至多一个来自 A_{i+1}，A_{i+2}，…，A_n 的实体组合相关联。因而联系 R 的主码可以用 A_1，A_2，…，A_i 的主码的并集来构造。
- 对每个实体集 A_k，$i<k\leqslant n$，来自其他实体集的实体的每个组合可以和来自 A_k 的至多一个实体相关联。于是对于 $i<k\leqslant n$，每个集合{A_1，A_2，…，A_{k-1}，A_{k+1}，…，A_n}构成一个候选码。

这两种解释在不同的书和系统中使用。为了避免混淆，我们只允许在一个联系集外有一个箭头，在这种情况下这两种解释是等价的。在第 8 章中(8.4 节)我们学习函数依赖，它允许以一种不会混淆的方式描述这两种解释。

7.5.6　弱实体集

考虑一个 *section* 实体，它由课程编号、学期、学年以及开课编号唯一标识。显然，开课实体和课程实体相关联。假定我们在实体集 *section* 和 *course* 之间创建了一个联系集 *sec_course*。

现在，发现 *sec_course* 中的信息是冗余的，由于 *section* 已有属性 *course_id*，它标识该开课所关联的课程。消除这种冗余的一个方法是删除联系 *sec_course*；然而，这么做使得 *section* 和 *course* 之间的联系隐含于一个属性中，这并不是我们想要的。 279

消除这种冗余的另一个方法是在实体 *section* 中不保存属性 *course_id*，而只保存剩下的属性 *sec_id*、

year 以及 *semester* ⊖。然而，这样的话实体集 *section* 就没有足够的属性唯一标识一个指定的 *section* 实体；即使每个 *section* 实体都是唯一的，不同课程的开课也可能会有相同的 *sec_id*、*year* 以及 *semester*。为解决这个问题，我们将联系 *sec_course* 视为一个特殊的联系，它给唯一标识 *section* 实体提供额外信息，即 *course_id*。

弱实体集的概念对上述想法进行了正式的规定。没有足够的属性以形成主码的实体集称作**弱实体集**(weak entity set)。有主码的实体集称作**强实体集**(strong entity set)。

弱实体集必须与另一个称作**标识**(identifying)或**属主实体集**(owner entity set)的实体集关联才能有意义。每个弱实体必须和一个标识实体关联；也就是说，弱实体集**存在依赖**(existence dependent)于标识实体集。我们称标识实体集**拥有**(own)它所标识的弱实体集。将弱实体集与其标识实体集相联的联系称为**标识性联系**(identifying relationship)。

标识性联系是从弱实体集到标识实体集多对一的，并且弱实体集在联系中的参与是全部的。标识性联系集不应该有任何描述性属性，因为这种属性中的任意一个都可以与弱实体集相关联。

在我们的例子中，*section* 的标识实体集是 *course*，将 *section* 实体和它们对应的 *course* 实体关联在一起的 *sec_course* 是标识性联系。

虽然弱实体集没有主码，但是我们仍需要区分依赖于特定强实体集的弱实体集中的实体的方法。弱实体集的**分辨符**(discriminator)是使得我们进行这种区分的属性集合。例如，弱实体集 *section* 的分辨符由属性 *sec_id*、*year* 以及 *semester* 组成，因为对每门课程来说，这个属性集唯一标识了这门课程的一次开课。弱实体集的分辨符也称为该实体集的**部分码**。

弱实体集的主码由标识实体集的主码加上该弱实体集的分辨符构成。在实体集 *section* 的例子中，它的主码是{*course_id*，*sec_id*，*year*，*semester*}，其中 *course_id* 是标识实体集 *course* 的主码，{*sec_id*，*year*，*semester*}区分同一门课程的不同 *section* 实体。

注意，我们可以选择使 *sec_id* 对于大学所提供的所有课程都不重复，在这种情况下实体集 *section*
280 将会具有一个主码。然而，一个 *section* 的存在在概念上仍依赖于一个 *course*，通过使之成为弱实体集可以明确这种依赖关系。

在 E-R 图中，弱实体集和强实体集类似，以矩形表示，但是有两点主要的区别：

- 弱实体集的分辨符以虚下划线标明，而不是实线。
- 关联弱实体集和标识性强实体集的联系集以双菱形表示。

在图 7-14 中，弱实体集 *section* 通过联系集 *sec_course* 依赖于强实体集 *course*。

图 7-14 包含弱实体集的 E-R 图

该图还表明了如何使用双线表明全部参与；(弱)实体集 *section* 在联系 *sec_course* 中的参与是全部的，表示每次开课都必须通过 *sec_course* 同某门课程关联。最后，从 *sec_course* 指向 *course* 的箭头表示每次开课与单门课程相关联。

弱实体集可以参与标识性联系以外的其他联系。例如，*section* 实体可以和 *time_slot* 实体集参与一个联系，以标识开课的时间。弱实体集可以作为属主与另一个弱实体集参与一个标识性联系。一个弱实体集也可能与不止一个标识实体集关联。这样，一个特定的弱实体将被一个实体的组合标识，其中每个标识实体集有一个实体在该组合中。弱实体集的主码可以由标识实体集的主码的并集加上弱实体集的分辨符组成。

在某些情况下，数据库设计者会选择将一个弱实体集表示为属主实体集的一个多值复合属性。在

⊖ 注意，即使我们从实体集 *section* 中去掉了 *course_id* 属性，但因为后面会看清楚的一些原因，我们最终从实体集 *section* 构建的关系模式还是具有 *course_id* 属性的。

我们的例子中，这种方法需要实体集 *course* 具有一个多值复合属性 *section*。如果弱实体集只参与标识性联系，而且其属性不多，那么在建模时将其表示为一个属性更恰当。相反地，如果弱实体集参与到标识性联系以外的联系中，或者其属性较多，则建模时将其表示为弱实体集更恰当。很明显，*section* 不 281
符合建模成多值复合属性的要求，而将其建模为弱实体集更恰当。

7.5.7　大学的 E-R 图

图 7-15 展示了本书迄今所使用的大学所对应的 E-R 图。除了增加了若干约束，以及 *section* 为弱实体以外，该 E-R 图与我们在 7.4 节中看到的大学 E-R 模型的文字性描述等价。

在我们的大学数据库中，我们限制每名教师必须有且仅有一个相关联的系。因此，如图 7-15 所示，在 *instructor* 和 *inst_dept* 之间有一条双线，表示 *instructor* 在 *inst_dept* 中全部参与；即每名教师必须和一个系相关联。另外，存在一个从 *inst_dept* 到 *department* 的箭头，表示每个教师可以有至多一个相关联的系。

类似地，实体集 *course* 和 *student* 分别与联系集 *course_dept* 和 *stud_dept* 用双线连接，实体集 *section* 和联系集 *sec_time_slot* 亦如此。前面的两个联系分别通过箭头指向另一个联系 *department*，而第三个联系通过箭头指向 *time_slot*。

此外，图 7-15 显示联系集 *takes* 具有一个描述性属性 *grade*，以及每个学生有至多一位导师。该图还表明目前 *section* 是一个弱实体集，由属性 *sec_id*、*semester* 以及 *year* 组成分辨符；*sec_course* 是连接弱实体集 *section* 和强实体集 *course* 的标识性联系集。

7.6 节将介绍 E-R 图能够如何用于转换为我们使用的各个关系模式。

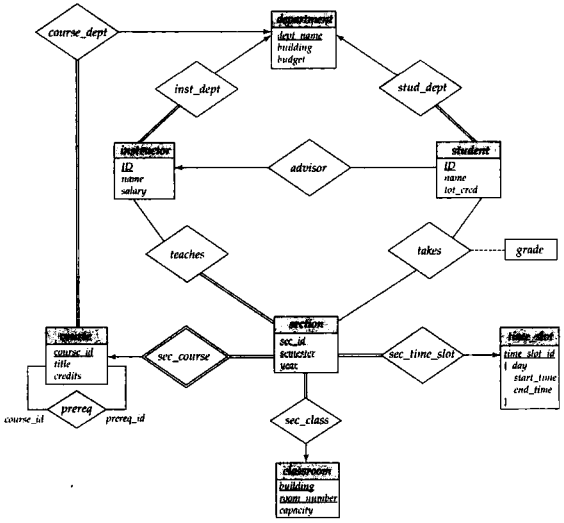


图 7-15　大学的 E-R 图

7.6　转换为关系模式

我们可以将一个符合 E-R 数据库模式的数据库表示为一些关系模式的集合。在数据库设计中，对于每个实体集以及对于每个联系集，都有唯一的关系模式与之对应，关系模式名即为相应的实体集或联系集的名称。

E-R 模型和关系数据库模型都是现实世界企业抽象的逻辑表示。由于两种模型采用类似的设计原

则，因此我们可以将 E-R 设计转换为关系设计。

这一节描述如何用关系模式来表示 E-R 模式，以及如何将 E-R 设计中提出的约束映射到关系模式上的约束。

7.6.1 具有简单属性的强实体集的表示

设 E 是只具有简单描述性属性 a_1，a_2，…，a_n 的强实体集。我们用具有 n 个不同属性的模式 E 来表示这个实体集。该模式的关系中的每个元组同实体集 E 的一个实体相对应。

对于从强实体集转换而来的模式，强实体集的主码就是生成的模式的主码。这个结论是从每个元组都对应于实体集中的一个特定实体这个事实直接得到的。

例如，考虑图 7-15 中 E-R 图的实体集 *student*。该实体集有三个属性：*ID*、*name* 和 *tot_cred*。我们用名为 *student* 的模式来表示这个实体集，它有三个属性：

$$student(\underline{ID},\ name,\ tot\_cred)$$

注意，由于学生 *ID* 是实体集的主码，因此它也是该关系模式的主码。

282
283

继续我们的例子，对于图 7-15 中的 E-R 图，除 *time_slot* 以外的所有的强实体集只有简单属性，从这些强实体集转换而来的模式为：

$$classroom(\underline{building},\ \underline{room\_number},\ capacity)$$
$$department(\underline{dept\_name},\ building,\ budget)$$
$$course(\underline{course\_id},\ title,\ credits)$$
$$instructor(\underline{ID},\ name,\ salary)$$
$$student(\underline{ID},\ name,\ tot\_cred)$$

如你所见，模式 *instructor* 和 *student* 与前面章节中用到的模式不同（它们不包含属性 *dept_name*）。后面很快还会讨论这个问题。

7.6.2 具有复杂属性的强实体集的表示

当一个强实体集具有非简单属性时，事情更加复杂一点。我们通过为每个子属性创建一个单独的属性来处理复合属性；我们并不为复合属性自身创建一个单独的属性。例如，考虑图 7-11 中表示的 *instructor* 实体集。对于复合属性 *name*，为 *instructor* 生成的模式包括属性 *first_name*、*middle_initial* 和 *last_name*；没有单独的属性或模式表示 *name*。类似地，对于复合属性 *address*，产生的模式包括属性 *street*、*city*、*state* 以及 *zip_code*。由于 *street* 是一个复合属性，因此它被替换成 *street_number*、*street_name* 以及 *apt_number*。8.2 节将再次讨论这个问题。

多值属性的处理不同于其他属性。我们已经看到，E-R 图中的属性通常都可以直接映射到相应关系模式的属性上。但是，多值属性是个例外；如我们将看到的，为了这些属性，创建新的关系模式。

派生的属性并不在关系数据模型中显式地表示出来。然而，它们可以在例如对象 - 关系数据模型的其他数据模型中表示为“方法”，第 22 章将对此进行讲述。

从具有复杂属性的实体集 *instructor* 转换而来，且不包含多值属性的关系模式为：

$$instructor(ID,\ first\_name,\ middle\_initial,\ last\_name,$$
$$street\_number,\ street\_name,\ apt\_number,$$
$$city,\ state,\ zip\_code,\ date\_of\_birth)$$

对于一个多值属性 M，构建关系模式 R，该模式包含一个对应于 M 的属性 A，以及对应于 M 所在的实体集或联系集的主码的属性。

例如，考虑图 7-11 中的 E-R 图，它描绘了包含多值属性 *phone_number* 的实体集 *instructor*。*instructor* 的主码是 *ID*。为这个多值属性构建一个关系模式：

284

$$instructor\_phone(\underline{ID},\ \underline{phone\_number})$$

教师的每个电话号码都表示为该模式上的关系中的唯一一个元组。因此，如果有一个 *ID* 为 22222 的教师，电话号码为 555 - 1234 和 555 - 4321，关系 *instructor_phone* 将有两条元组（22222，555 - 1234）和（22222，555 - 4321）。

创建关系模式的主码，它由模式中的所有属性组成。在上面的例子中，主码由关系 *instructor_phone*

的两个属性一起组成。

另外，在多值属性构建的关系模式上建立外码约束，由实体集的主码所生成的属性去参照实体集所生成的关系。在上面的例子中，*instructor_phone* 关系上的外码约束是属性 *ID* 参照 *instructor* 关系。

在一个实体集只有两个属性的情况下——一个主码 B 以及一个多值属性 M——该实体集的关系模式只包含一个属性，即主码属性 B。可以删掉这个关系，同时保留具有属性 B 和对应 M 的属性 A 的关系模式。

例如，考虑图 7-15 中描绘的实体集 *time_slot*，其中 *time_slot_id* 是实体集 *time_slot* 的主码，有一个多值属性，而且恰好是复合的。这个实体集可以就按照以下从多值复合属性生成的模式表示：

$$time\_slot(\underline{time\_slot\_id}, \underline{day}, \underline{start\_time}, end\_time)$$

虽然没有在 E-R 图中表示为约束，但是我们知道不存在一个班的两次课在一周中的同一天的同一时间开始却在不同时间结束。基于这个约束，*end_time* 已从模式 *time_slot* 的主码中去除了。

从实体集生成的关系将只有一个属性 *time_slot_id*，去掉此关系的优化有助于简化生成的数据库模式，即使它有一个与外码相关的缺点，7.6.4 节将对此简要讨论。

7.6.3　弱实体集的表示

设 A 是具有属性 a_1，a_2，…，a_m 的弱实体集，设 B 是 A 所依赖的强实体集，设 B 的主码包括属性 b_1，b_2，…，b_n。我们用名为 A 的关系模式表示实体集 A，该模式的每个属性对应以下集合中的一个成员：

$$\{a_1, a_2, \cdots, a_m\} \cup \{b_1, b_2, \cdots, b_n\}$$

对于从弱实体集转换而来的模式，该模式的主码由其所依赖的强实体集的主码与弱实体集的分辨符组合而成。除了创建主码之外，还要在关系 A 上建立外码约束，该约束指明属性 b_1，b_2，…，b_n 参照关系 B 的主码。外码约束保证表示弱实体的每个元组都有一个表示相应强实体的元组与之对应。 285

以图 7-15 所示 E-R 图中的弱实体集 *section* 为例说明。该实体集有属性：*sec_id*、*semester* 和 *year*。*section* 实体集所依赖的实体集 *course* 的主码是 *course_id*。因此，用来表示 *section* 的模式具有下面的属性：

$$section(\underline{course\_id}, \underline{sec\_id}, \underline{semester}, \underline{year})$$

该主码由实体集 *course* 的主码和 *section* 的分辨符（即 *sec_id*、*semester* 以及 *year*）组成。我们还在模式 *section* 上建立了一个属性 *course_id* 参照 *course* 模式的主码的外码约束，以及完整性约束“级联删除”[⊖]。由于外码约束上的“级联删除”规范，如果一个 *course* 实体被删除，那么所有与它相关联的 *section* 实体也被删除。

7.6.4　联系集的表示

设 R 是联系集，设 a_1，a_2，…，a_m 表示所有参与 R 的实体集的主码的并集构成的属性集合，设 R 的描述性属性（如果有）为 b_1，b_2，…，b_n。我们用名为 R 的关系模式表示该联系集，下面集合中的每一项表示为模式的一个属性：

$$\{a_1, a_2, \cdots, a_m\} \cup \{b_1, b_2, \cdots, b_n\}$$

7.3.3 节介绍过如何为一个二元联系集选择主码。如在该节中我们所看到的，从所有相关实体集中取所有主码属性能够用来标识一个指定元组，但是对于一对一、多对一和一对多联系集，这会得到一个比我们所需要的主码大的属性集合。因而如下选择主码：

- 对于多对多的二元联系，参与实体集的主码属性的并集成为主码。
- 对于一对一的二元联系集，任何一个实体集的主码都可以选作主码。这个选择是任意的。
- 对于多对一或一对多的二元联系集，联系集中“多”的那一方的实体集的主码构成主码。 286

⊖ 4.4.5 节介绍了 SQL 中外码约束的“级联删除”性质。

- 对于边上没有箭头的 n 元联系集，所有参与实体集的主码属性的并集成为主码。
- 对于边上有一个箭头的 n 元联系集，不在“箭头”侧的实体集的主码属性为模式的主码。回想一下，一个联系集外只允许一个箭头。

我们还在关系模式 R 上建立外码约束，如下：对于每个与联系集 R 相关的实体集 E_i，我们建立一个关系模式 R 上的外码约束，R 中来自 E_i 主码属性的那些属性参照表示关系模式 E_i 的主码。

以图 7-15 中 E-R 图的联系集 *advisor* 为例说明。此联系集涉及如下两个实体集：

- *instructor*，主码为 *ID*。
- *student*，主码为 *ID*。

由于该联系集没有属性，因此 *advisor* 模式有两个属性，*instructor* 和 *student* 的主码。由于这两个属性具有相同的名字，因此将它们重命名为 *i_ID* 和 *s_ID*。因为 *advisor* 联系集是从 *student* 到 *instructor* 多对一的，所以关系模式 *advisor* 的主码是 *s_ID*。

我们还在关系 *advisor* 上建立了两个外码约束，属性 *i_ID* 参照 *instructor* 的主码，属性 *s_ID* 参照 *student* 的主码。

继续我们的例子，对于图 7-15 的 E-R 图，从联系集派生的模式如图 7-16 所示。

观察到对于联系集 *prereq*，与联系相关联的角色标识用作属性的名字，这是因为两个角色都参照同一个关系 *course*。

与 *advisor* 的情况类似，*sec_course*、*sec_time_slot*、*secVclass*、*inst_dept*、*stud_dept* 以及 *course_dept* 每个关系的主码仅由两个相关联的实体集中的一个实体集的主码构成，因为每个对应的联系都是多对一的。

图 7-16 中并没有表示出外码，但是对于该图中的每一个关系都有两个外码约束，参照相关的两个实体集所构建出的两个关系。例如，*sec_course* 有参照 *section* 和 *classroom* 的外码，*teaches* 有参照 *instructor* 和 *section* 的外码，以及 *takes* 有参照 *student* 和 *section* 的外码。

teaches (*ID*, *course_id*, *sec_id*, *semester*, *year*)
takes (*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
prereq (*course_id*, *prereq_id*)
advisor (*s_ID*, *i_ID*)
sec_course (*course_id*, *sec_id*, *semester*, *year*)
sec_time_slot (*course_id*, *sec_id*, *semester*, *year*, *time_slot_id*)
sec_class (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*)
inst_dept (*ID*, *dept_name*)
stud_dept (*ID*, *dept_name*)
course_dept (*course_id*, *dept_name*)

图 7-16 由图 7-15 中 E-R 图的联系集派生出的模式

这个优化允许对包含多值属性的实体集 *time_slot* 只建立一个关系模式，而避免从关系模式 *sec_time_slot* 到由实体集 *time_slot* 生成的关系的外码的建立，因为我们舍弃了由实体集 *time_slot* 生成的关系。我们保留了从多值属性生成的名为 *time_slot* 的关系，但这个关系可能没有元组与某个 *time_slot_id* 相关联，或者有多条元组与某个 *time_slot_id* 相关联，因而 sec_time 中的 *time_slot_id* 不能参照这个关系。

精明的读者可能会想，为什么我们在此前的章节中没有见过模式 *sec_course*、*sec_time_slot*、*sec_class*、*inst_dept*、*stud_dept* 以及 *course_dept*。原因是我们迄今所提出的算法使得一些模式或者被消除，或者和其他模式合并。我们接下来讨论这个问题。

7.6.4.1 模式的冗余

连接弱实体集和相应强实体集的联系集比较特殊。如 7.5.6 节提到的，这样的联系集是多对一的，且没有描述性属性。另外，弱实体集的主码包含强实体集的主码。在图 7-14 的 E-R 图中，弱实体集 *section* 通过联系集 *sec_course* 依赖于强实体集 *course*。*section* 的主码是 {*course_id*, *sec_id*, *semester*, *year*}，*course* 的主码是 *course_id*。由于 *sec_course* 没有描述性属性，因此 *sec_course* 模式有属性 *course_id*、*sec_id*、*semester* 以及 *year*。表示实体集 *section* 的模式包含属性 *course_id*、*sec_id*、*semester* 以及 *year* 等。*sec_course* 关系中的每个(*course_id*, *sec_id*, *semester*, *year*)组合也将出现在 *section* 模式上的关系中，反之亦然。因此，*sec_course* 模式是冗余的。

一般情况下，连接弱实体集与其所依赖的强实体集的联系集的模式是冗余的，而且在基于 E-R 图的关系数据库设计中不必给出。

7.6.4.2　模式的合并

287 | 288

考虑从实体集 *A* 到实体集 *B* 的一个多对一的联系集 *AB*。用前面讲的关系 - 模式构建算法，得到三个模式：*A*、*B* 和 *AB*。进一步假设 *A* 在该联系中的参与是全部的；即实体集 *A* 中的每个实体 *a* 都必须参与到联系 *AB* 中。那么我们可以将 *A* 和 *AB* 模式合并成单个包含两个模式所有属性的并集的模式。合并后模式的主码是其模式中融入了联系集模式的那个实体集的主码。

让我们检验图 7-15 的 E-R 图中满足上述条件的关系以讲解：

- *inst_dept*。模式 *instructor* 和 *department* 分别对应于实体集 *A* 和 *B*。因此模式 *inst_dept* 可以和模式 *instructor* 合并。结果是 *instructor* 模式由属性 {*ID*, *name*, *dept_name*, *salary*} 组成。
- *stud_dept*。模式 *student* 和 *department* 分别对应于实体集 *A* 和 *B*。因此模式 *stud_dept* 可以和模式 *student* 合并。结果是 *student* 模式由属性 {*ID*, *name*, *dept_name*, *tot_cred*} 组成。
- *course_dept*。模式 *course* 和 *department* 分别对应于实体集 *A* 和 *B*。因此模式 *course_dept* 可以和模式 *course* 合并。结果是 *course* 模式由属性 {*course_id*, *title*, *dept_name*, *credits*} 组成。
- *sec_class*。模式 *section* 和 *classroom* 分别对应于实体集 *A* 和 *B*。因此模式 *sec_class* 可以和模式 *section* 合并。结果是 *section* 模式由属性 {*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*} 组成。
- *sec_time_slot*。模式 *section* 和 *time_slot* 分别对应于实体集 *A* 和 *B*。因此模式 *sec_time_slot* 可以和上一步中得到的模式 *section* 合并。结果是 *section* 模式由属性 {*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*} 组成。

在一对一的联系的情况下，联系集的关系模式可以跟参与联系的任何一个实体集的模式进行合并。

即使参与是部分的，我们也可以通过使用空值来进行模式的合并。在上面这个例子中，如果 *inst_dept* 是部分参与的，那么我们可以为那些没有相关联的系的教师在属性 *dept_name* 中存放空值。

最后，我们考虑表示联系集的模式上本应有的外码约束。参照每一个参与联系集的实体集的外码约束本应存在。我们舍弃了参照联系集模式所合并入的实体集模式的约束，然后将另一个外码约束加到合并的模式中。在上面的例子中，*inst_dept* 上有一个 *dept_name* 属性参照 *department* 关系的外码约束，当模式 *inst_dept* 与 *instructor* 合并时，这个外码约束被加到 *instructor* 关系中。

289

7.7　实体 - 联系设计问题

实体集和联系集的标记法并不精确，而且定义一组实体及它们的相互联系可能有多种不同的方式。本节讨论 E-R 数据库模式设计中的一些基本问题。设计过程将在 7.10 节更详细地讨论。

7.7.1　用实体集还是用属性

考虑具有新增属性 *phone_number* 的实体集 *instructor*（见图 7-17a）。显然电话可以作为一个单独的实体，具有属性 *phone_number* 和 *location*；地点可以是电话所处于的办公室或家，移动电话则可以用值“移动”来表示。如果我们采用这样的观点，那么我们就不给 *instructor* 增加属性 *phone_number*，反之，我们创建：

- 实体集 *phone*，具有属性 *phone_number* 和 *location*。
- 联系集 *inst_phone*，表示教师与他们所拥有的电话之间的关联。

这种方法如图 7-17b 所示。

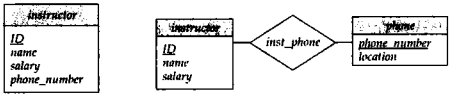


图 7-17　给 *instructor* 实体集增加 *phone* 的两种方法

那么，教师的这两个定义之间的主要差别是什么呢？将电话看成一个属性 *phone_number* 暗示每个

教师恰好有一个电话号码与之相关联。将电话看成一个实体 *phone*，允许每个教师可以有若干个电话号码(包括零个)与之相关联。然而，也可以简单地将 *phone_number* 定义为多值属性，从而允许每个教师有多个电话。

那么，主要的差别是，在一个人可能希望保存关于电话的额外信息，如它的位置，或类型(移动的、视频的或普通的老式电话)，或共享该电话的所有的人时，将电话看作一个实体是一种更好的建模方式。因此，把电话视为一个实体比把它视为一个属性的方式更具通用性；而且当通用性可能有用的时候，这种定义方式就更为适合了。

相反，将(一名教师的)*name* 属性视作一个实体就不太合适；将 *name* 说成是一个实体本身就很不
290 具有说服力(与电话相反)。因此，恰当的做法是将 *name* 视作 *instructor* 实体集的一个属性。

由此自然就产生两个问题：什么构成属性？什么构成实体集？很遗憾，对这两个问题并不能简单地回答。区分它们主要依赖于被建模的现实世界的企业的结构，以及被讨论的属性的相关语义。

一个常见的错误是用一个实体集的主码作为另一个实体集的属性，而不是用联系。例如，即使每名教师只指导一名学生，将 *student* 的 *ID* 作为 *instructor* 的属性也是不正确的。用 *advisor* 联系代表学生和教师之间的关联才是正确的方法，因为这样可以明确地表示出两者之间的关系而不是将这种关系隐含在属性中。

人们常犯的与此相关的另一个错误是将相关实体集的主码属性作为联系集的属性。例如，*ID*(*student* 的主码属性)和 *ID*(*instructor* 的主码)不应该在 *advisor* 联系中作为属性出现。这样做是不对的，因为在联系集中已经隐含了这些主码属性[⊖]。

7.7.2 用实体集还是用联系集

一个对象最好被表述为实体集还是联系集并不总是显而易见的。如图 7-15 所示，我们用 *takes* 联系集对学生选择课程(的某一次开课)建模。另一种方法是想像对于每个学生选的每门课程有一个课程－注册记录。那么，我们用一个叫作 *registration* 的实体集代表课程－注册记录。每个 *registration* 实体恰好与一个学生和一次开课相关联，因此我们有两个联系集，一个将课程－注册记录和学生关联，另一个将课程－注册记录和课程关联。如图 7-18 所示，我们将图 7-15 中 section 和 student 实体集之间的 *takes* 联系集用一个实体集和两个联系集替代：

- *registration*，代表课程－注册记录的实体集。
- *section_reg*，关联 *registration* 和 *course* 的联系集。
- *student_reg*，关联 *registration* 和 *student* 的联系集。

291 注意，我们使用双线表示 *registration* 实体全部参与。

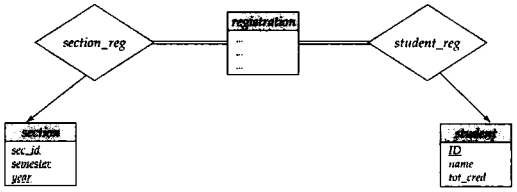


图 7-18 用 *registration* 和两个联系集替代 *takes*

图 7-15 和图 7-18 的方法都准确表达了大学的信息，但是使用 *takes* 的方法更紧凑也更可取。然而，如果注册办公室通过课程－注册记录与其他信息相关联，那么最好让它本身为一个实体。

在决定用实体集还是联系集时可采用的一个原则是，当描述发生在实体间的行为时采用联系集。这一方法在决定是否将某些属性表示为联系可能更适合时也很有用。

⊖ 我们以后会看到，当从 E-R 模式中创建关系模式时，这些属性可能会出现在从 *advisor* 联系集创建出的模式中，但是，它们并不应该出现在 *advisor* 联系集中。

7.7.3　二元还是 n 元联系集

数据库中的联系通常都是二元的。一些看来非二元的联系实际上可以用多个二元联系更好地表示。例如，可以创建一个三元联系 *parent*，将一个孩子与他/她的母亲和父亲相关联。然而，这一联系也可以用两个二元联系分别来表示，即 *mother* 和 *father*，分别将孩子与他/她的母亲和父亲相关联。使用 *mother* 和 *father* 两个联系使我们可以记录孩子的母亲，即使我们不知道父亲是谁；而对于这种情况三元联系 *parent* 中必须有一个空值。所以在这个例子中用二元联系更好。

事实上，一个非二元的(n 元，$n>2$)联系集总可以用一组不同的二元联系集来替代。简单起见，考虑一个抽象的三元($n=3$)联系集 R，它将实体集 A、B 和 C 联系起来。用实体集 E 替代联系集 R，并创建三个联系集，如图 7-19 所示：

- R_A，关联 E 和 A。
- R_B，关联 E 和 B。
- R_C，关联 E 和 C。

292

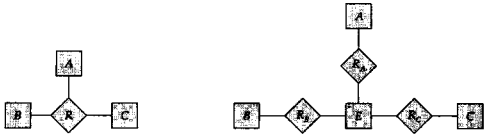


图 7-19　三元联系与三个二元联系

如果联系集 R 有属性，那么将这些属性赋给实体集 E；进一步，为 E 创建一个特殊的标识属性(因为它必须能够通过其属性值来区别实体集中的各个实体)。针对联系集 R 中的每个联系(a_i, b_i, c_i)，在实体集 E 中创建一个新的实体 e_i。然后，在三个新联系集中，分别插入新联系如下：

- 在 R_A 中插入(e_i, a_i)。
- 在 R_B 中插入(e_i, b_i)。
- 在 R_C 中插入(e_i, c_i)。

可以将这一过程直接推广到 n 元联系集的情况。因此，在概念上可以限制 E-R 模型只包含二元联系集。然而，这种限制并不总是令人满意的。

- 对于为表示联系集而创建的实体集，我们可能不得不为其创建一个标识属性。该标识属性和额外所需的那些联系集增加了设计的复杂程度以及对总的存储空间的需求(我们将在 7.6 节看到这一点)。
- n 元联系集可以更清晰地表示几个实体集参与单个联系集。
- 有可能无法将三元联系上的约束转变为二元联系上的约束。例如，考虑一个约束，表明 R 是从 A、B 到 C 多对一的；也就是，来自 A 和 B 的每一对实体最多与一个 C 实体关联。这种约束就不能用联系集 R_A、R_B 和 R_C 上的基数约束来表示。

考虑 7.2.2 节中的联系集 *proj_guide*，它关联 *instructor*、*student* 和 *project*。不能直接将 *proj_guide* 拆分为 *instructor* 和 *project* 之间的二元联系和 *student* 和 *project* 之间的二元联系。如果这么做，可以记录教 293
师 Katz 同学生 Shankar 和 Zhang 一起参与项目 A 和 B；然而无法记录 Katz 同 Shankar 一起参与项目 A 并且同 Zhang 一起参与项目 B，而不是同 Zhang 一起参与项目 A 或者同 Shankar 一起参与项目 B。

联系集 *proj_guide* 可以通过创建一个如上所述的新实体集来拆分为二元联系。然而，这么做却不是很自然。

7.7.4　联系属性的布局

一个联系的映射基数比率会影响联系属性的布局。因此，一对一或一对多联系集的属性可以放到一个参与该联系的实体集中，而不是放到联系集中。例如，我们指明 *advisor* 是一个一对多的联系集，也就是一个教师可以指导多个学生，但每个学生只能有一个导师。在这种情况下，表示教师何时成为

学生导师的属性 *date* 可以与 *student* 实体集相关联，如图 7-20 所示。(为了保持图例简单，只显示了两个实体集的部分属性。)由于每个 *student* 实体最多和一个 *instructor* 实例相关联，因此将属性 *date* 放在 *student* 实体集中和将属性 *date* 放在 *advisor* 联系集中具有相同的含义。一对多联系集的属性仅可以重置到参与联系的"多"方的实体集中。而对于一对一的联系集，联系的属性可以放到任意一个参与联系的实体中。

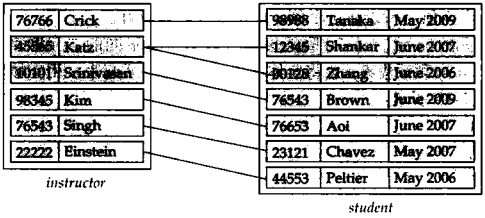


图 7-20 *date* 作为 *student* 实体集的属性

设计时将描述性属性作为联系集的属性还是实体集的属性这一决定应该反映出被建模企业的特点。
294 设计者可以选择保留 *date* 作为 *advisor* 的属性，以显式地表明指导关系的日期，而不是学生校内状态的其他一些方面(例如，被大学录取的日期)。

属性位置的选择在多对多联系集中体现得更清楚。回到刚才的例子，让我们指出可能更符合实际的情况，定义 *advisor* 为一个多对多的联系集，表明一个教师可以指导一个或多个学生，而一个学生可以被一个或多个教师指导。如果想要表示一个特定的教师成为一个特定学生的导师的日期，*date* 则必须作为联系集 *advisor* 的属性，而不是任何一个参与的实体集的属性。例如，如果将 *date* 作为 *student* 的属性，则我们无法知道哪个教师在该特定日期成为他的导师。当一个属性是由参与的实体集联合确定而不是由单独的某个实体集确定时，该属性就必须放到多对多联系集中。图 7-3 给出了作为联系属性时 *date* 的位置。为了图例的简单，只显示了两个实体集的部分属性。

7.8 扩展的 E-R 特性

虽然基本的 E-R 概念已足以对大多数数据库特征建模，但数据库的某些方面可以通过对基本 E-R 模型作某些扩展来更恰当地表述。这一节将讨论以下扩展 E-R 特性：特化、概化、高层和低层实体集、属性继承和聚集。

为了有助于讨论，我们将用一个稍微更复杂的大学数据库模式。特别是，我们将通过定义具有属性 *ID*、*name* 以及 *address* 的实体集 *person* 来对学校中不同的人建模。

7.8.1 特化

实体集可能包含一些子集，子集中的实体在某些方面区别于实体集中的其他实体。例如，实体集中的某个实体子集可能具有不被该实体集中所有实体所共享的一些属性。E-R 模型提供了表示这种与众不同的实体组(子集)的方法。

例如，实体集 *person* 可进一步归类为以下两类之一：

- *employee*。
- *student*。

这两个类中的每一个都用一个属性集来描述，包括实体集 *person* 的所有属性加上可能的附加属性。例如，*employee* 实体可进一步用属性 *salary* 来描述，而 *student* 实体可进一步用 *tot_cred* 属性来描述。在实体集内部进行分组的过程称为**特化**(specialization)。*person* 的特化使得我们可以根据他们是雇员还是学
295 生来区分人：一般来说，一个人可以是一个雇员、一个学生，都是，或者都不是。

另一个例子，假设大学希望将学生分为两类，研究生和本科生。给研究生配备办公室，而给本科生安排宿舍。每一个学生类型都通过属性集来描述，这个属性集包括 *student* 实体集的所有属性和附加

的属性。

大学可以创建 *student* 的两个特化，*graduate* 和 *undergraduate*。如我们此前所看到的，学生实体用属性 *ID*、*name*、*address* 以及 *tot_cred* 描述。实体集 *graduate* 将具有 *student* 的所有属性以及一个附加属性 *office_number*。实体集 *undergraduate* 将具有 *student* 的所有属性以及一个附加属性 *residential_college*。

我们可以不断重复地使用特化来完善设计。例如，大学雇员可进一步划分为以下两类之一：

- *instructor*。
- *secretary*。

每类雇员都用包括实体集 *employee* 的所有属性以及附加属性的属性集来描述。例如，*instructor* 实体可以进一步由属性 *rank* 来描述，而 *secretary* 实体可以由属性 *hours_per_week* 来描述。进一步，*secretary* 实体可以参与实体集 *secretary* 和 *employee* 之间的 *secretary_for* 联系，它标识了有秘书协助的雇员。

一个实体集可以根据多个可区分的特征进行特化。在我们的例子中，雇员实体间的可区分特征是雇员所从事的工作。同时，另一个特化可以基于一个人是临时(有限任期)雇员还是长期雇员，从而有实体集 *temporary_employee* 和 *permanent_employee*。当一个实体集上形成了多于一种特化时，某个特定实体可能同时属于多个特化实体集。例如，一个特定的雇员可以既是一个临时的雇员，又是一个秘书。

在 E-R 图中，特化用从特化实体指向另一方实体的空心箭头来表示(如图 7-21 所示)。我们称这种关系为 ISA 关系，它代表“is a”，表示“是一个”，例如，一个教师“是一个”雇员。

我们在 E-R 图中描述特化的方法取决于一个实体集是否可能属于多个特化实体集或者它是否必须属于至多一个特化实体集。前者(允许多个集)称为**重叠特化**(overlapping specialization)，后者(允许至多一个)称为**不相交特化**(disjoint specialization)。对于一个重叠特化(例如 *student* 和 *employee* 作为 *person* 的特化的情况)，分开使用两个箭头。对于一个不相交特化(例如 *instructor* 和 *secretary* 作为 *employee* 的特化的情况)，使用一个箭头。特化关系还可能形成**超类 - 子类**(superclass-subclass)联系。高层和低层实体集按普通实体集表示——即包含实体集名称的矩形。

7.8.2　概化

从初始实体集到一系列不同层次的实体子集的细化代表了一个**自顶向下**(top-down)的设计过程，在这个设计过程中，显式地产生出差别。设计过程也可以**自底向上**(bottom-up)进行，多个实体集根据共同具有的特征综合成一个较高层的实体集。数据库设计者可能一开始就标识了：

- *instructor* 实体集，具有属性 *instructor_id*、*instructor_name*、*instructor_salary* 以及 *rank*。
- *secretary* 实体集，具有属性 *secretary_id*、*secretary_name*、*secretary_salary* 以及 *house_per_week*。

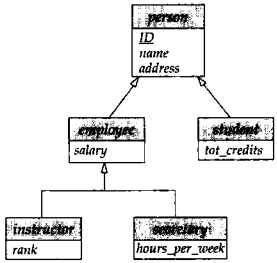


图 7-21　特化和概化

就所具有的属性而言，在 *instructor* 实体集和 *secretary* 实体集间存在共性。从概念上来说，这两个实体集包含相同的属性：也就是标识符、姓名和工资属性。这种共性可以通过**概化**(generalization)来表达，概化是高层实体集与一个或多个低层实体集间的包含关系。在我们的例子中，*employee* 是高层实体集，而 *instructor* 和 *secretary* 是低层实体集。在这种情况下，在两个低层实体集中，概念上相同的属性有不同的名字。为了进行概化，这些属性必须赋予相同的名字并由高层实体 *person* 表示。我们使用 *ID*、*name* 和 *address* 作为属性名称，正如我们在 7.8.1 节中所看到的那样。

296
/
297

高层与低层实体集也可以分别称作**超类**(superclass)和**子类**(subclass)。实体集 *person* 是子类 *employee* 和 *student* 的超类。

对于所有实际应用来说，概化只不过是特化的逆过程。为企业设计 E-R 模型时，我们将配合使用这两个过程。在 E-R 图中，我们对概化和特化的表示不作区分。为了使设计模式完全体现数据库应用和数据库用户的要求，我们会通过区分(特化)或综合(概化)来产生新的实体层次。这两种方式的区别主要在于它们的出发点和总体目标。

特化从单一的实体集出发，通过创建不同的低层实体集来强调同一实体集中不同实体间的差异。低层实体集可以有不适用于高层实体集中所有实体的属性，也可以参与到不适用于高层实体集中所有实体的联系中。设计者采用特化的原因正是为了表达这种与众不同的特征。如果 *student* 和 *employee* 与 *person* 实体拥有完全相同的属性，并且与 *person* 实体参与完全相同的联系，则没有必要特化 *person* 实体集。

概化的进行基于这样的认识：一定数量的实体集共享一些共同的特征(即用相同的属性描述它们，且它们都参与到相同的联系集中)。概化是在这些实体集的共性的基础上将它们综合成一个高层实体集。概化用于强调低层实体集间的相似性并隐藏它们的差异；由于共享属性的不重复出现，它还使得表达简洁。

7.8.3 属性继承

由特化和概化所产生的高层和低层实体的一个重要特性是**属性继承**(attribute inheritance)。高层实体集的属性被低层实体集**继承**(inherit)。例如，*student* 和 *employee* 继承了 *person* 的属性。因此，*student* 用属性 *ID*、*name* 和 *address* 以及附加属性 *tot_cred* 来描述；而 *employee* 用属性 *ID*、*name* 和 *address* 以及附加属性 *salary* 来描述。属性继承适用于所有低层实体集，因此 *employee* 的子类 *instructor* 和 *secretary* 从 *person* 继承了属性 *ID*、*name* 和 *address*，另外又从 *employee* 继承了 *salary*。

低层实体集(或子类)同时还继承地参与其高层实体(或超类)所参与的联系集。和属性继承类似，参与继承适用于所有低层实体集。例如，假设实体集 *person* 和 *department* 参与 *person_dept* 联系集。那么，实体集 *person* 的子类实体集 *student*、*employee*、*instructor* 和 *secretary* 也都和 *department* 隐式地参与到
298 *person_dept* 联系中。以上这些实体集可以参与到 *person* 实体参与的任何联系中。

对 E-R 图的一个给定的部分来说，不管它是通过特化还是通过概化得到的，其结果都是一样的：

- 高层实体集所关联的所有属性和联系适用于它的所有低层实体集。
- 低层实体集特有的性质仅适用于特定的低层实体集。

在后面的叙述中，虽然我们常常只说概化，但我们所讨论的性质是两个过程所共有的。

图 7-21 所示为实体集的**层次结构**(hierarchy)。在图 7-21 中，*employee* 是 *person* 的低层实体集，同时又是 *instructor* 和 *secretary* 实体集的高层实体集。在层次结构中，给定的实体集作为低层实体集只参与到一个 ISA 联系中，即在这个图中实体集只具有**单继承**(single inheritance)。如果一个实体集作为低层实体集参与到多个 ISA 联系中，则称这个实体集具有**多继承**(multiple inheritance)，且产生的结构被称为**格**(lattice)。

7.8.4 概化上的约束

为了更准确地对企业建模，数据库设计者可能选择在特定概化上设置某些约束。一类约束包含判定哪些实体能成为给定低层实体集的成员。成员资格可以是下列中的一种：

- **条件定义的**(condition-defined)。在条件定义的低层实体集中，成员资格的确定基于实体是否满足一个显式的条件或谓词。例如，假设高层实体集 *student* 具有属性 *student_type*。所有 *student* 实体都根据 *student_type* 属性进行评估。只有满足条件 *student_type* = "研究生"的实体才允许属于 *graduate_student* 低层实体集。所有满足条件 *student_type* = "本科生"的实体都包含于 *undergraduate_student*。由于所有低层实体都基于同一属性(在这里基于 *student_type*)进行评估，因此这种类型的概化称作是属性定义的(attribute-defined)。
- **用户定义的**(user-defined)。用户定义的低层实体集不是通过成员资格条件来限制，而是由数据库用户将实体指派给某个实体集。例如，如果我们假设大学雇员在 3 个月的雇佣期后被分配到 4 个工作组中的一个。因此我们用高层实体集 *employee* 的 4 个低层实体集来表示工作组。一个给定的员工并不是根据某个明确定义的条件自动分配到某个特定工作组实体中。反而是负责决策的用户根据个人观点进行工作组的分配。该分派是通过将一个实体加入某个实体集的操作而
299 实现的。

另一类约束涉及在一个概化中一个实体是否可以属于多个低层实体集。低层实体集可能是下述情况之一：

- **不相交**(disjoint)。不相交约束要求一个实体至多属于一个低层实体集。在我们的例子中，*student* 实体在 *student - type* 属性上只能满足一个条件；一个实体可以是研究生或本科生，但不能既是研究生又是本科生。
- **重叠**(overlapping)。在重叠概化中，同一个实体可以同时属于同一个概化中的多个低层实体集。我们再来看员工工作组的例子，并假设某些雇员参加到多个工作组中。在这种情况下给定雇员就可以出现在 *employee* 的多个低层工作组实体集中。因此这种概化是重叠的。

在图 7-21 中，我们假定一个人可以既是雇员又是学生。我们通过分开的箭头表示这种重叠概化：一个箭头从 employee 指向 person，另一个箭头从 student 指向 person。然而，instructor 和 secretary 的概化是不相交的，我们对此用单个箭头表示。

最后一类约束是对概化的**完全性约束**(completeness constraint)，定义高层实体集中的一个实体是否必须至少属于该概化/特化的一个低层实体集。这种约束可以是下述情况之一：

- **全部概化**(total generalization)或**特化**(specialization)。每个高层实体必须属于一个低层实体集。
- **部分概化**(partial generalization)或**特化**(specialization)。允许一些高层实体不属于任何低层实体集。

部分概化是默认的。我们可以在 E-R 图中表示全部概化，即通过在图中加入关键词“total”，并画一条从关键词到相应的空心箭头(表示不相交概化)的虚线，或者画一条到空心箭头集合(表示重叠概化)的虚线。

student 的概化是全部的：所有的学生实体都要么是研究生要么本科生。由于通过概化产生的高层实体集通常只包括低层实体集中的实体，因此由概化产生的高层实体集的完全性约束通常是全部的。如果概化是部分的，高层实体就可以不限制出现在任何低层实体集中。工作组实体集就是部分特化的例子。由于员工在工作 3 个月后才分配到某个工作组中，因此某些 *employee* 实体可能不属于任何低层工作组实体集。

我们可以通过对 *employee* 的部分的、重叠的特化过程来更完全地表示出工作组实体集的特征。从 *graduate_student* 和 *undergraduate_student* 到 *student* 的概化是全部的、不相交的概化。然而，完全性约束 300
和不相交约束彼此并没有依赖关系。约束模式也可以是部分 - 不相交或全部 - 重叠的。

我们可以看到，对给定概化或特化使用约束带来某些插入和删除需求。例如，当存在一个全部的完全性约束时，插入到高层实体集中的实体还必须插入到至少一个低层实体集中。在条件定义的约束中，所有满足条件的高层实体必须插入到相应的低层实体集中。最后，从高层实体集删除的实体也必须从它所属于的所有相应低层实体集中删除。

7.8.5　聚集

E-R 模型的一个局限性在于它不能表达联系间的联系。为了说明这种结构的必要性，考虑我们此前看到的 *instructor*、*student* 和 *project* 之间的三元联系 *proj_guide*(如图 7-13 所示)。

现在假设每位在项目上指导学生的教师需要记录月评估报告。我们将评估报告建模为一个主码为 *evaluation_id* 的实体 *evaluation*。记录一个 *evaluation* 对应的(*student*、*project*、*instructor*)组合的另一个方法是在 *instructor*、*student*、*project* 和 *evaluation* 之间建立一个四元(四路)联系集 *eval_for*。(四元的联系是必需的——例如，*student* 和 *evaluation* 之间的二元联系无法让我们表示某个 *evaluation* 对应的(*project*，*instructor*)组

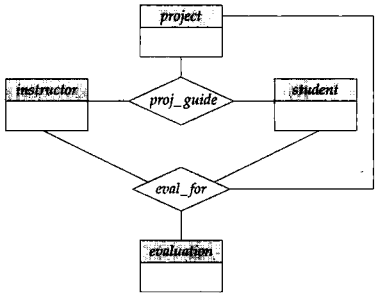


图 7-22　包含冗余联系的 E-R 图

合)。通过使用基本的 E-R 模型构建，我们得到了图 7-22 所示的 E-R 图。(为了简洁，省略了实体集的属性。)

看上去联系集 *proj_guide* 和 *eval_for* 可以合并到一个联系集中。然而，我们不应该将它们合并到一起，因为一些 *instructor*、*student*、*project* 组合可能没有相关联的 *evaluation*。

但是，按照这种方法产生的图存在冗余信息，因为在 *eval_for* 中的每个 *instructor*、*student*、*project* 组合肯定也在 *proj_guide* 中。如果 *evaluation* 是一个值而不是一个实体，我们可以将 *evaluation* 作为联系集 *proj_guide* 的一个多值属性。然而，当某个 *evaluation* 和其他实体相关联时，这种方法不可行；例如，每份评估报告可能和负责评估报告的后续处理以发放奖学金的 *secretary* 相关联。

对类似上述情况建模的最好办法是使用聚集。**聚集**(aggregation)是一种抽象，通过这种抽象，联系被视为高层实体。这样，对于我们的例子，我们将联系集 *proj_guide*(关联实体集 *instructor*、*student* 和 *project*)看成一个名为 *proj_guide* 的高层实体集。这种实体集可以像对任何其他实体集一样来处理。这样我们就可以在 *proj_guide* 和 *evaluation* 之间创建一个二元联系 *eval_for* 来表示某个 *evaluation* 对应哪个(*student*, *projec*, *instructor*)组合。图 7-23 所示为用聚集概念来表示以上这种情况。

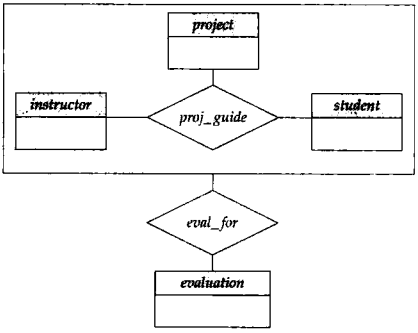


图 7-23 包含聚集的 E-R 图

7.8.6 转换为关系模式

我们现在开始介绍扩展的 E-R 特性如何转换为关系模式。

7.8.6.1 概化的表示

为包含概化的 E-R 图进行关系模式设计有两种不同方法。尽管我们在下面的讨论中考虑图 7-21 的概化的情况，但为了简化讨论，我们谈到的内容实际上只包括第一层的低层实体集——*employee* 和 *student*。我们假定 *ID* 是 *person* 的主码。

- 为高层实体集创建一个模式。为每个低层实体集创建一个模式，模式中的属性包括对应于低层实体集的每个属性，以及对应于高层实体集主码的每个属性。因此，对于图 7-21 的 E-R 图(忽略 *instructor* 和 *secretary* 实体集)，有三个模式：

  person(*ID*, *name*, *street*, *city*)
  employee(*ID*, *salary*)
  student(*ID*, *tot_cred*)

  高层实体集的主码属性变成既是高层实体集的主码属性也是所有低层实体集的主码属性。如上例中下划线标记所示。

  另外，我们在低层实体集上建立外码约束，其主码属性参照创建自高层实体集的关系的主码。在上面的例子中，*employee* 的属性 *ID* 会参照 *person* 的主码，对于 *student* 类似。

- 如果概化是不相交且完全的——如果不存在同时属于两个同级的低层实体集的实体，且如果高层实体集的任何实体也都是某个低层实体集的成员——那么可以采用另一种表示方法。这时，我们不需要为高层实体集创建任何模式，只需要为每个低层实体集创建一个模式，模式中的属性包括对应于低层实体集的每个属性，以及对应于高层实体集的每个属性。那么，对于图 7-21 的 E-R 图，有两个模式：

  employee(*ID*, *name*, *street*, *city*, *salary*)
  student(*ID*, *name*, *street*, *city*, *tot_cred*)

301 ~ 303

这两个模式都将高层实体集 *person* 的主码属性 *ID* 作为它们的主码。

第二种方法的一个缺点在于定义外码约束。为了说明这个问题，假定我们有一个与实体集 *person*

相关的联系集 R。用第一种方法，当从该联系集创建一个关系模式 R 时，也可以在 R 上建立参照 *person* 模式的外码约束。遗憾的是，如果用第二种方法，R 上的外码约束无法参照单一的一个关系。为了避免这个问题，需要创建一个关系模式 *person*，该模式至少包含实体 *person* 的主码属性。

如果将第二种方法用于重叠概化，某些值就会不必要地存储多次。例如，如果一个人既是雇员又是学生，*street* 和 *city* 的值就会存储两次。

如果概化是不相交的但不是全部的——有的人既不是雇员又不是学生——则将需要一个额外的模式

$$person(\underline{ID}, name, street, city)$$

来表示这样的人。然而，上述外码约束问题仍然存在。为了解决这个问题，假定在 *person* 关系中额外表示了雇员和学生，遗憾的是，姓名、街道以及城市的信息就要在 *person* 关系和 *student* 关系中冗余地存储，同样，雇员的信息在 *person* 关系和 *employee* 关系中冗余地存储。这就促使将姓名、街道和城市信息只存储在 *person* 关系中，而把这些信息从 *student* 和 *employee* 中移除。如果我们这么做的话，结果就跟我们此前讲述的第一种方法完全一样了。

7.8.6.2　聚集的表示

为包含聚集的 E-R 图进行模式设计是很直接的。考虑图 7-23 中的 E-R 图。表示聚集 *proj_guide* 和实体集 *evaluation* 之间联系的联系集 *eval_for* 的模式包含对应实体集 *evaluation* 主码中的每个属性以及联系集 *proj_guide* 主码中的每个属性。它还包含对应于联系集 *eval_for* 的任意描述性属性（如果存在的话）。然后，根据我们已经定义的规则，在聚集的实体集中转换联系集和实体集。

当把聚集像其他实体集一样看待时，我们此前看到的用于在联系集上创建主码和外码约束的规则，也同样可以应用于与聚集相关联的联系集。聚集的主码是定义该聚集的联系集的主码。不需要单独的关系来表示聚集；而使用从定义该聚集的联系创建出来的关系就可以了。

7.9　数据建模的其他表示法

一个应用的数据模型的图形表示对于数据库模式的设计至关重要。数据库模式的构建不仅需要数据建模专家，而且还需要了解应用的需求但可能并不熟悉数据建模的领域专家。一个直观的图形表示使两类专家间的信息沟通变得简单，因而尤其重要。 304

目前已经提出了许多种对数据建模的表示法，其中 E-R 图和 UML 类图的应用最为广泛。对于 E-R 表示法还没有一个统一的标准，不同的书籍以及 E-R 图软件使用不同的表示法。我们在本书第 6 版中选择了一种特定的表示法，它不同于本书以往版本使用的表示法，我们将在本节的稍后解释原因。

在本节剩下的部分中，我们学习一些可选择的 E-R 图表示法，以及 UML 类图表示法。为了将我们的表示法与其他表示法比较，图 7-24 总结了我们此前在 E-R 图表示法中用到的符号集。

7.9.1　E-R 图的其他表示法

图 7-25 列出一部分广泛使用的可选择的 E-R 表示法。表示实体的属性的一种可选的方法是将它们放入与代表实体的方框所连接的椭圆形中。主码属性以下划线表明。在图 7-25 的最上部展示了上述表示法。联系的属性也可以用类似的方式表示，即将包含属性的椭圆与表示联系的菱形连接起来。

如图 7-25 所示，联系上的基数约束可以用多种不同的方法表示。在图 7-25 的左部展示了一种可选择的方法，即在联系外面的边上标记 * 和 1，通常用来表示多对多、一对一和多对一联系。一对多的情况与多对一的情况是对称的，在此没有画出。

在图 7-25 的右部展示了另一种可选择的表示法，联系集是用实体集之间的连线而不是菱形来表示；因此只有二元联系才可以如此表示。如图 7-25 所示，在这种表示法中基数约束用“鸦爪形”表示法表示。在 $E1$ 和 $E2$ 之间的联系 R 中，两侧的鸦爪表示 $E1$ 到 $E2$ 的多对多的关系。在这个表示法中，竖线代表全部参与。不过要注意，在实体 $E1$ 和 $E2$ 间的联系 R 中，如果 $E1$ 对 R 全部参与，则竖线放置在对面，即靠近 $E2$ 的位置；类似地，在对面画一个圆圈表示部分参与。

图 7-25 的底部是概化的另一种可选择的表示方法，即用三角形替代空心箭头。

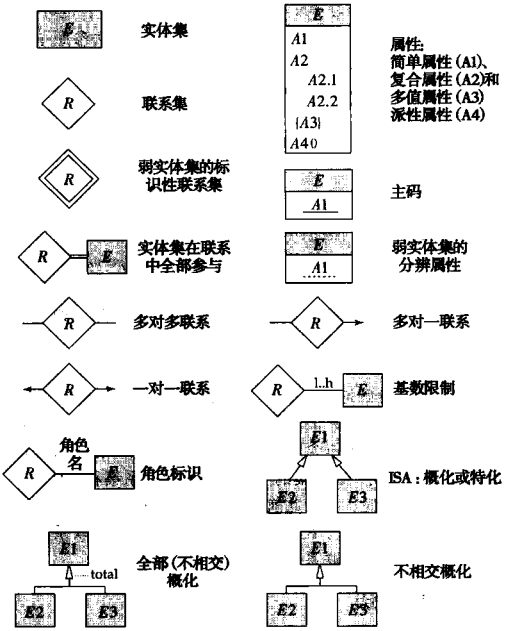


图 7-24 E-R 图表示法中使用的符号

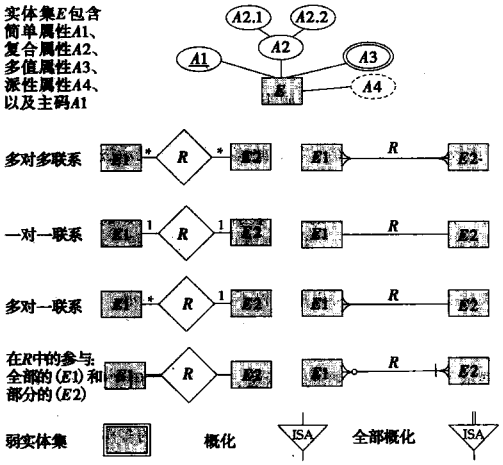


图 7-25 其他可选择的 E-R 图表示法

在本书第 5 版及以前的版本中，我们用椭圆形表示属性，用三角形表示概化，如图 7-25 所示。用

椭圆形表示属性并用菱形表示联系的表示法接近于 Chen 在他引入 E-R 模型概念的论文中所用的 E-R 图的原始形式，那种表示法目前称为陈氏表示法。

美国国家标准和技术研究院于 1993 年定义了一个称为 IDEF1X 的标准，其中使用了鸦爪形符号并在联系的边上加上竖线用于指明全部参与，加上空心圆表示部分参与，还包括了其他没有列出的符号。

随着统一标记语言(Unified Markup Language，UML)(将在 7.9.2 节介绍)使用的增多，我们选择更新 E-R 表示法，使之更接近 UML 类图的形式；7.9.2 节将说明它们之间的联系。和此前的表示法相比，新的表示法对属性的表示更加紧凑，并且更接近许多 E-R 建模工具所支持的表示法，此外也更接近 UML 类图表示法。

目前有多种用于构建 E-R 图的工具，每个工具都有它自己的表示法变体。其中的一些工具甚至提供在几种不同的 E-R 表示法变体间的选择。更多信息请参考文献注解部分的引用文献。

在 E-R 图中的实体集与从这些实体集创建的关系模式之间一个关键的不同点在于，E-R 联系对应的关系模式中的属性，比如 *instructor* 的属性 *dept_name*，在 E-R 图中的实体集中并没有表示出来。一些数据建模工具允许用户在同一个实体的两种视图间选择，一种是不包含这些属性的实体视图，而另一种是包含这些属性的关系视图。

305 ~ 308

7.9.2　统一建模语言 UML

实体 - 联系图有助于对软件系统的数据表示部分建模。然而，数据表示只构成整个系统设计的一部分。其他部分包括系统用户界面的建模、系统功能模块的规范定义以及它们之间的交互等。**统一建模语言**(Unified Modeling Language，UML)是由对象管理组织(Object Management Group，OMG)主持开发的一个标准，它是为了建立软件系统不同部分的规范定义而提出的。UML 的一些组成部分为：

- **类图**(class diagram)。类图和 E-R 图相类似。本节将说明类图的一些特征及其与 E-R 图的关系。
- **用况图**(use case diagram)。用况图说明了用户和系统之间的交互，特别是用户所执行的任务中的每一步操作(如取钱或注册课程)。
- **活动图**(activity diagram)。活动图说明了系统不同部分之间的任务流。
- **实现图**(implementation diagram)。实现图在软件构件层和硬件构件层说明了系统的各组成部分以及它们之间的联系。

在这里我们不准备提供 UML 各部分的细节。关于 UML 的参考文献请参看文献注解。不过，我们将通过一些例子来说明 UML 中与数据建模有关的部分的一些特征。

图 7-26 显示了几个 E-R 图的构造和与它们等价的 UML 类图的构造。我们将在下面描述这些构造。事实上 UML 为对象建模，而 E-R 为实体建模。对象和实体很像，也有属性，但是另外还提供一组函数(称为方法)，它们可在对象属性的基础上调用以计算值，或更新对象本身。类图除了可以说明属性外，还可以说明方法。我们在第 22 章讨论对象。UML 不支持复合或多值属性，派生属性与不带参数的函数等价。由于类支持封装，因此 UML 允许属性和函数带有前缀“+”、“-”或“#”，这分别表示公共、私有以及受保护的访问。私有属性只能在类的方法中使用，而受保护的属性只能在类和它的子类的方法中使用；了解 Java、C ++ 或 C#的人应该对此很熟悉。

在 UML 术语中，联系集称为**关联**(association)；为了与 E-R 术语一致，我们将仍称它们为联系集。在 UML 中我们通过划一条线段连接实体集来表示二元联系集。我们将联系集的名称写在线段的附近。我们还可以通过将角色的名称写在靠近实体集的线段上，来说明联系集中一个实体集的角色。另一种方法是，我们可将联系集的名字写在方框里，和联系集的属性写在一起，并用虚线把这个方框连接到表示联系集的连线上。这个方框可以看作一个实体集，如同 E-R 图中的聚集一样，也可以与其他实体集一起参与联系。

从 UML 1.3 版开始，UML 通过使用与 E-R 图中使用的一样的菱形表示法支持非二元联系。而在更早版本的 UML 中无法直接表示非二元联系——非二元联系必须用我们在 7.7.3 节中看见的技术转换成二元联系。即使对于二元联系，UML 也允许使用菱形表示法，但是大部分设计者使用线段表示法。

基数约束在 UML 中和 E-R 图中一样用 $l..h$ 的形式表示，其中 l 表示参与联系的实体的最小个数，h 则表示最大个数。然而，如图 7-26 所示，你应该注意到约束的位置和在 E-R 图中约束的位置正好相

反。*E*2 边上的约束 0..* 和 *E*1 边上的 0..1 表示每个 *E*2 实体可以参与至多一个联系，而每个 *E*1 实体可以参与很多联系；换句话说，该联系是从 *E*2 到 *E*1 多对一的。

像 1 或 * 这样的单个值可以写在边上；边上的单个值 1 视为与 1..1 等价，而 * 等价于 0.. *。UML 支持概化，表示法与 E-R 表示法基本相同，包括不相交概化和重叠概化的表示方式。

UML 类图还包含一些其他的表示法，与我们见过的 E-R 表示法并不对应。例如，连接两个实体集的一条线的一端有一个小菱形，表示菱形这一端的实体集包含另一个实体集(包含关系在 UML 术语中称为“聚合”，不要将聚合的这种用法同 E-R 模型中聚集的含义混淆)。例如，一个车辆实体可能包含一个发动机实体。

UML 类图还提供了表示面向对象语言的特征的表示法，例如接口。关于 UML 类图的更多信息，请参看文献注解中的引用文献。

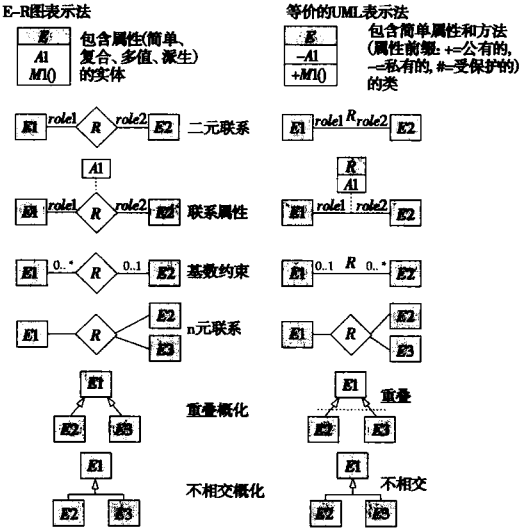


图 7-26 UML 类图表示法中使用的符号

7.10 数据库设计的其他方面

本章对模式设计的扩展讨论可能会造成模式设计是数据库设计的唯一组成部分的错误印象。我们将在后续章节中更完整地讲述一些其他的考虑，而在此，我们先简要地概览一下。

7.10.1 数据约束和关系数据库设计

我们已经看到，使用 SQL 可以表达多种数据约束，包括主码约束、外码约束、check 约束、断言和触发器。约束用于多种目的。最明显的一个目的是自动的一致性保持。通过在 SQL 数据定义语言中表达约束，设计者能够确保数据库系统自己执行这些约束。这比让每个应用程序自己独立地执行约束要可靠得多。同时，这种机制也为更新和添加约束提供了一个集中的位置。

显式声明约束的另一个优点是一些约束在关系数据库模式的设计中特别有用。例如，如果我们知道社会保障号唯一地标识一个人，那么我们就可以使用一个人的社会保障号来关联与这个人相关的数据，即使这些数据出现在多个关系中。与此相反，比如，眼睛的颜色不是唯一标识符。眼睛的颜色不能用于关联与某个人相关的多个关系中的数据，因为这样就无法将一个人的数据同其他具有同样眼睛颜色的人的相关数据区分开来。

在 7.6 节中，通过使用设计中指定的约束，我们为一个给定的 E-R 设计生成了一个关系模式集合。在第 8 章中，我们将这个思想及其相关约束形式化，并展示它将如何辅助关系数据库模式的设计。关系数据库设计的规范化方法使我们能够在给定的设计是好的设计时用准确的方式进行表述，并且能够把不好的设计转变为较好的设计。我们将看到，这个从实体 - 联系设计开始并根据该设计规则地生成关系模式的过程给整个设计过程提供了一个好的开始。 309 / 310

数据约束在确定数据的物理结构时同样有用，可以将彼此紧密相关的数据存储在磁盘上邻近的区域，以便在磁盘访问时提高效率。当索引建立在主码上时，索引结构工作得更好。

每次数据库更新时，执行约束会在性能上带来潜在的高代价。对于每次更新，系统都必须检查所有的约束，然后要么拒绝与约束冲突的更新，要么运行相应的触发器。性能损失的严重性不仅仅取决于更新的频率，而且依赖于数据库的设计方式。对某些类型的约束进行检测的真实效率，是第 8 章中对关系数据库模式设计的讨论的一个重要方面。

7.10.2　使用需求：查询、性能

数据库系统的性能是绝大多数企业信息系统的一个关键因素。性能不仅与计算能力的有效利用以及所使用的存储硬件有关，而且受到与系统交互的人的效率以及依赖数据库数据的处理的效率的影响。

以下是效率的两个主要度量方法：

- **吞吐量**(throughput)——每单位时间里能够处理的查询或更新(通常指事务)的平均数量。
- **响应时间**(response time)——单个事务从开始到结束所需的平均时间或者最长时间。

以批量的方式处理大量事务的系统关注于达到高吞吐量。与人交互或者时间苛刻的系统则通常关注于响应时间。这两个度量并不等价。高吞吐量的目的是为了获得系统部件的高利用率。这样做的时候可能会导致某些事务延迟，直到它们能更高效地运行的时候才处理。所以这些延迟的事务的响应时间就很差。

大多数商用数据库系统长期以来都关注于吞吐量，但是，包括基于 Web 的应用和电信信息系统等在内的许多应用都要求好的平均响应时间和适当限制内的最差响应时间。

了解预期最频繁的查询的类型有助于设计过程。涉及连接的查询比不涉及连接的查询需要更多的资源用以计算。在需要连接的情况下，数据库管理员可以选择创建一个索引，加快连接的计算。对于查询——不论是否涉及连接——可以创建索引以加速常常出现在查询中的选择谓词(SQL 的 **where** 子句)的计算。查询中另一个影响索引选择的因素是更新和读操作的混合。当一个索引可能加速查询的同时，它也可能减缓更新的速度，因为更新会为维护索引的准确性而强制性地带来额外的工作。 311

7.10.3　授权需求

授权约束同样会影响数据库的设计，因为 SQL 允许在数据库逻辑设计组件的基础上将访问权限授予用户。一个关系模式可能需要分解为两个或多个模式，以便于在 SQL 中授予访问权限。比如，一个雇员记录可以包括与工资单、工作职责和医疗保险相关的数据。由于企业的不同管理部门分别管理该数据的不同部分，有些用户需要访问工资数据，但却不能访问工作数据和医疗数据等。如果这些数据在一个表中，希望得到的访问划分通过使用视图尽管依然可行，但更加麻烦。当数据在一个计算机网络中的多个系统中分散存放时，这种方式的数据划分则是必不可少的，这个问题将在第 19 章讨论。

7.10.4　数据流、工作流

数据库应用通常是大型企业应用的一部分，这样的应用不仅与数据库系统交互，而且会同各种专门的应用交互。例如，在一个制造公司中，计算机辅助设计(CAD)系统会用于辅助新产品的设计。CAD 系统可以通过 SQL 语句从数据库中抽取数据，在其内部处理这些数据，也许同时和产品设计人员进行交互，然后再更新数据库。在这个过程中，数据的控制权会在几个产品设计人员和其他人之间传递。再看一个例子，考虑一个差旅费报告。它由一个出差归来的雇员写成(可能利用某个专门的软件包)，然后依次交给该雇员的经理，可能的其他高层经理，最后交到了财务部门以报销费用(在此处它将与该企业的财务信息系统交互)。

术语*工作流*表示一个流程中的数据和任务的组合，前面给出了这种流程的两个例子。当工作流在

用户间移动以及用户执行他们在工作流中的任务时，工作流会与数据库系统交互。除了工作流操作的数据之外，数据库还可以存储工作流自身的数据，包括构成工作流的任务以及它们在用户之间移动的路径。因此工作流可以列出一系列对数据库的查询和更新，而这些可能会作为数据库设计过程的一个部分考虑进来。换句话说，对企业建模要求我们不仅要理解数据的语义，还要理解使用这些数据的业
312 务流程。

7.10.5 数据库设计的其他问题

数据库设计通常不是一个一蹴而就的工作。一个组织的需求不断发展，它所需要存储的数据也会相应地发展。在最初的数据库设计阶段，或者在应用的开发中，数据库设计者都有可能意识到在概念、逻辑和物理模式层次上要有所改变。模式上的改变会影响到数据库应用的方方面面。一个好的数据库设计会预先估计一个组织将来的需要，设计出的模式在需求发展时只需要做最少的改动即可满足要求。

区分预期持久的基本约束和预期要改变的约束非常重要。例如，教师编号能唯一地标识一名教师的约束是基本的。另一方面，大学可能有一项规定，一名教师只能属于一个系，这条规定也许会在将来改变，如果允许联合任命的话。只允许每个教师属于一个系的数据库设计在允许联合任命时会需要较大的改动。只要每个教师只有一个主系，这种联合任命就可以通过新添一个关系来表示，而不需要修改 *instructor* 关系。而如果规定改成允许一个教师有多个主系时，则数据库设计就需要进行较大的改动。一个好的设计应该不止考虑当前的规定，还应该避免或者最小化由预计或有可能发生的改变而带来的改动。

进一步说，数据库所服务的企业很可能会与其他企业交互，因此，多个数据库可能需要进行交互。不同模式之间的数据转换在现实世界的应用中是一个重要的问题。对于这个问题提出了很多解决方案。我们将在第 23 章学习的 XML 数据模型，在不同应用间交换数据时，广泛用于表示数据。

最后，不言而喻，数据库设计在两个意义上是面向人的工作：系统的最终用户是人(即使有应用程序位于数据库和最终用户之间)；数据库设计者需要与应用领域的专家进行广泛交互以理解应用的数据需求。所有涉及数据的人都有需要和偏好，为了数据库设计和部署在企业中获得成功，这些都应当考虑到。

7.11 总结

- 数据库设计主要涉及数据库模式的设计。**实体 - 联系**(Entity-Relationship，E-R)数据模型是一个广泛用于数据库设计的数据模型。它提供了一个方便的图形化表示方法以查看数据、联系和约束。
313
- E-R 模型主要用于数据库设计过程。它的发展是为了帮助数据库设计，这是通过允许定义**企业模式**(enterprise schema)实现的。这种企业模式代表数据库的全局逻辑结构，该全局结构可以用 **E-R 图**(E-R diagram)图形化地表示。
- **实体**(entity)是在现实世界中存在并且区别于其他对象的对象。我们通过把每个实体同描述该实体的一组属性相关联来表示区别。
- **联系**(relationship)是多个实体间的关联。相同类型的联系的集合为**联系集**(relationship set)，相同类型的实体的集合为**实体集**(entity set)。
- 术语**超码**(superkey)、**候选码**(candidate key)以及**主码**(primary key)同适用于关系模式一样适用于实体和联系集。在确定一个联系集的主码时需要小心，因为它由来自一个或多个相关的实体集的属性组成。
- **映射的基数**(mapping cardinality)表示通过联系集可以和另一实体相关联的实体的个数。
- 不具有足够属性构成主码的实体集称为**弱实体集**(weak entity set)。具有主码的实体集称为**强实体集**(strong entity set)。
- E-R 模型的各种性质为数据库设计者提供了大量的选择，使设计人员可以最好地表示被建模的企业。在某些情况中，概念和对象可以用实体、联系或属性来表示。企业总体结构的各方面可以用弱实体集、概化、特化或聚集很好地描述。设计者通常需要在简单的、紧凑的模型与更精确但也更复杂的模型之间进行权衡。

- 用 E-R 图定义的数据库设计可以用关系模式的集合来表示。数据库的每个实体集和联系集都有唯一的关系模式与之对应，其名称即为相应的实体集或联系集的名称。这是从 E-R 图转换为关系数据库设计的基础。
- **特化**(specialization)和**概化**(generalization)定义了一个高层实体集和一个或多个低层实体集之间的包含关系。特化是取出高层实体集的一个子集来形成一个低层实体集。概化是用两个或多个不相交的(低层)实体集的并集形成一个高层实体集。高层实体集的属性被低层实体集继承。
- **聚集**(aggregation)是一种抽象，其中联系集(和跟它们相关的实体集一起)被看作高层实体集，并且可以参与联系。
- UML 是一种常用的建模语言。UML 类图广泛用于对类建模以及一般的数据建模。 314

术语回顾

- 实体 - 联系数据模型
- 实体和实体集
  - 属性
  - 域
  - 简单和复合属性
  - 单值和多值属性
  - 空值
  - 派生属性
- 超码、候选码以及主码
- 联系和联系集
  - 二元联系集
  - 联系集的度
  - 描述性属性
  - 超码、候选码以及主码
  - 角色
  - 自环联系集
- E-R 图
- 映射基数
  - 一对一联系
  - 一对多联系
  - 多对一联系
  - 多对多联系
- 参与
  - 全部参与
  - 部分参与
- 弱实体集和强实体集
  - 分辨符属性
  - 标识联系
- 特化和概化
  - 超类和子类
  - 属性继承
  - 单和多继承
  - 条件定义的和用户定义的成员资格
  - 不相交概化和重叠概化
  - 全部概化和部分概化
- 聚集
- UML
- UML 类图

实践习题

7.1　为车辆保险公司构建一个 E-R 图，它的每个客户有一辆或多辆车。每辆车关联零次或任意次事故的记录。每张保险单为一辆或多辆车保险，并与一个或多个保费支付相关联。每次支付只针对特定的一段时间，具有关联的到期日和缴费日。

7.2　考虑一个用于记录学生在不同开课(section)的不同考试中所得成绩的数据库。 315

- 为数据库构造一个 E-R 图，其中，将考试建模为实体并使用一个三元联系。
- 构造另一个 E-R 图，其中，只用 *students* 和 *section* 间的二元联系。保证在特定 *student* 和 *section* 对之间只存在一个联系；而且你可以表示出学生在不同考试中所得成绩。

7.3　设计一个 E-R 图用于跟踪记录你最喜欢的球队的成绩。你应该保存打过的比赛，每场比赛的比分，每场比赛的上场队员以及每个队员在每场比赛中的统计数据。总的统计数据应该被建模成派生属性。

7.4　考虑一个 E-R 图，其中相同实体集出现数次，且它的属性重复出现多次。为什么这样的冗余是应尽量避免的不良设计？

7.5　E-R 图可视为一个图。下面这些术语对于一个企业模式的结构意味着什么？

a. 图是非连通的。

b. 图是有环的。

7.6　如 7.7.3 节描述及图 7-27b 所示(属性没有列出)，考虑用多个二元联系来表示一个三元联系。

a. 给出 E、A、B、C、R_A、R_B 和 R_C 的一个简单实例，这个实例不能对应于 A、B、C 和 R 的任何实例。

b. 修改图 7-27b 的 E-R 图，引入约束，确保 E、A、B、C、R_A、R_B 和 R_C 的任意满足约束的实例将对应于 A、B、C 和 R 的一个实例。

c. 修改以上的转换以处理该三元联系上的全部参与约束。

d. 以上表示方式需要我们为 E 创建一个主码属性。试问如何将 E 看成弱实体集从而不需要主码属性？

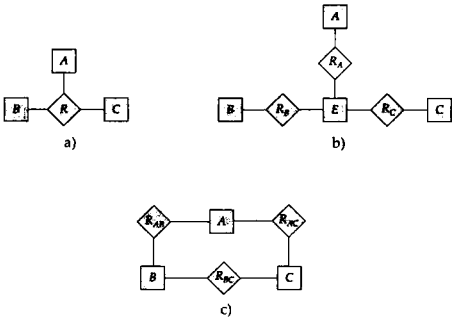


图 7-27 实践习题 7.6 和习题 7.24 的 E-R 图

7.7 一个弱实体集总可以通过往自己的属性中加入其标识实体集的主码属性而变成一个强实体集。概括一下如果我们这么做会产生什么样的冗余。

7.8 考虑一个关系，例如 *sec_course*，产生于多对一的联系 *sec_course*。在这个关系上创建的主码约束和外码约束是否强制实施多对一的基数约束？解释为什么。

7.9 假设 *advisor* 联系是一对一的。为了确保执行一对一基数约束，在关系 *advisor* 上需要哪些额外的约束？

7.10 考虑实体集 *A* 和 *B* 之间多对一的联系 *R*。假设从 *R* 生成的关系和 *A* 生成的关系合并了。在 SQL 中，参与外码约束的属性可以为空。解释如何使用 SQL 上的 not null 约束来强制实施 *A* 在 *R* 中的全部参与约束。

7.11 在 SQL 中，外码约束只能参照被参照关系的主码属性，或者其他用 unique 约束声明为超码的属性。这导致多对多联系（或者一对多联系的“一”方）上的全部参与约束在从该联系创建的关系上无法用关系上的主码、外码以及非空约束强制实施。

a. 解释为什么。

b. 解释如何使用复杂的 check 约束或断言（见 4.4.7 节）强制实施全部参与约束。（遗憾的是，在目前广泛使用的数据库中并不支持这些特性。）

7.12 图 7-28 所示为概化和特化的一个格结构（属性没有给出）。针对实体集 *A*、*B* 和 *C*，说明如何从高层实体集 *X* 和 *Y* 继承属性。讨论 *X* 的一个属性和 *Y* 的某个属性同名时应如何处理。

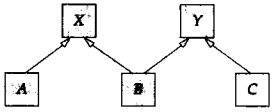


图 7-28 实践习题 7.12 的 E-R 图

7.13 **时间变化**（temporal change）：E-R 图通常对一个企业在某个时间点的状态建模。假设我们希望追踪时间变化，即数据随时间的变化。例如，张可能在 2005 年 9 月 1 日至 2009 年 5 月 31 日之间是一名学生，而 Shankar 在 2008 年 5 月 31 日至 2008 年 12 月 5 日以及 2009 年 6 月 1 日至 2010 年 1 月 5 日期间的导师是 Einstein。类似地，一个实体或联系的属性值会随时间发生变化，例如 *course* 的 *title* 和 *credits*，*instructor* 的 *salary* 甚至 *name*，以及 *student* 的 *tot_cred*。

对时间变化建模的一种方法如下。我们定义一个新的数据类型**有效时间**（valid_time），它是一个时间段或者时间段的集合。然后我们将每一个实体和联系都与一个 *valid_time* 属性关联起来，记录实体或联系有效的时间段。一个时间段的结束时间可以是无穷，例如，如果 Shankar 在 2008 年 9 月 2 日成为学生，并且目前仍为学生，我们可以将 Shankar 实体 *valid_time* 时间段的结束时间表示为无穷。类似地，我们将值随时间变化的属性建模为值的集合，每一个值具有自己的 *valid_time*。

a. 画一个包含 *student* 和 *instructor* 实体以及 *advisor* 联系并具有上述扩展以追踪时间变化的 E-R 图。

b. 将上面的 E-R 图转换为一个关系集合。

很明显，上面产生的关系集非常复杂，使得像写 SQL 语句这样的任务比较难完成。另一种使用得

比较多的方法是在设计 E-R 模型的时候忽略时间变化(尤其是属性值随时间变化)，然后修改由 E-R 模型生成的关系以追踪时间变化，8.9 节将对此进行讨论。 316 / 318

习题

7.14　解释主码、候选码和超码这些术语之间的区别。

7.15　为医院构建一个包含一组病人和一组医生的 E-R 图。为每个病人关联一组不同的检查和化验记录。

7.16　为实践习题 7.1 ~ 7.3 中的每个 E-R 图构建适当的关系模式。

7.17　扩展实践习题 7.3 中的 E-R 图，以追踪整个联赛中所有队伍的相同信息。

7.18　解释弱实体集和强实体集之间的区别。

7.19　我们可以通过简单地增加一些适当的属性，将任意弱实体集转变成强实体集。那么，我们为什么还要弱实体集呢?

7.20　考虑图 7-29 中的 E-R 图，它对一家网上书店建模。

a. 列出实体集及其主码。

b. 假设书店向其展览的商品中增加了蓝光光盘和可下载视频。相同的商品可能在一种格式中存在，或在两种格式中都存在且具有不同的价格。扩展 E-R 图来为这个新增需求建模，忽略对购物篮的影响。

- 现在用概化来扩展 E-R 图，从而对包含书、蓝光光盘或可下载视频的任意组合的购物篮进行建模。

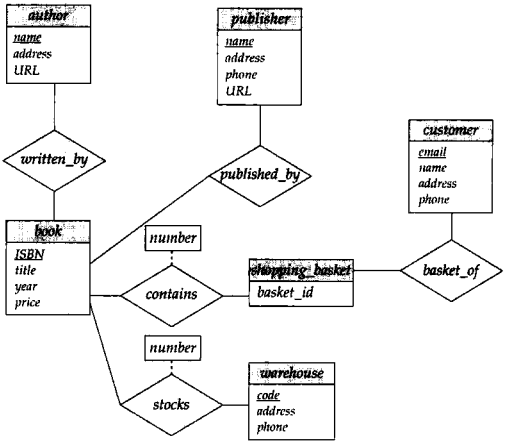


图 7-29　习题 7.20 的 E-R 图

7.21　为一个汽车公司设计一个数据库，用于协助它的经销商维护客户记录以及经销商库存，并协助销售人员订购车辆。

每辆车由车辆编号(Vehicle Identification Number，VIN)唯一标识，每辆单独的车都是公司提供的特定品牌的特定模型(例如，XF 是塔塔汽车捷豹品牌的模型)。每个模型都可以有不同的选项，但是一辆车可能只有一些(或没有)可用的选项。数据库需要保存关于模型、品牌、选项的信息，以及每个经销商、顾客和车的信息。

你的设计应该包括 E-R 图、关系模式的集合，以及包括主码约束和外码约束的一组约束。

7.22　为全球性的快递公司(例如 DHL 或者 FedEX)设计一个数据库。数据库必须能够追踪(寄件的)客户和(收件的)客户；有些客户可能两者都是。由于每个包裹必须是可标识且可追踪的，因此数据库必须能够存储包裹的位置信息以及它的历史位置。位置包括卡车、飞机、机场和仓库。 319

你的设计应该包括 E-R 图、关系模式的集合，以及包括主码约束和外码约束的一组约束。

7.23 为航空公司设计一个数据库。数据库必须追踪客户和他们的预订、航班、他们在航班上的状态、座位分配，以及未来航班的时刻表和飞行路线。

你的设计应该包括 E-R 图、关系模式的集合，以及包括主码约束和外码约束的一组约束。

7.24 在 7.7.3 节，我们用多个二元联系（如图 7-27b 所示）来表示一个三元联系（见图 7-27a）。考虑图 7-27c 中的另一种方式。讨论这两种用多个二元联系来表示一个三元联系的方式的相对优点。

320

7.25 考虑 7.6 节中所示从图 7-15 的 E-R 图转化而来的关系模式。对于每个模式，指出应该创建的外码约束（如果有的话）。

7.26 为机动车辆销售公司设计一个概化—特化层次结构。该公司出售摩托车、小客车、货车和大巴。论述你在层次结构的各层设定属性位置的合理性。说明为什么它们不应放在较高的层次或较低的层次。

7.27 说明条件定义的和用户定义的约束的差别。系统能自动检查哪种约束？解释你的答案。

7.28 说明不相交约束和重叠约束之间的区别。

7.29 解释全部约束和部分约束之间的区别。

工具

许多数据库系统提供支持 E-R 图的数据库设计工具。这些工具有助于设计者创建 E-R 图，并且它们可在数据库中自动创建相应的表。可参看第 1 章文献注解中给出的数据库系统厂商的网址。

还存在一些独立于数据库并且支持 E-R 图和 UML 类图的数据建模工具。画图工具 Dia 是一款免费软件，支持 E-R 图和 UML 类图。商用工具包括 IBM Rational Rose（www. ibm. com/software/rational）、Microsoft Visio（见 www. microsoft. com/office/visio）、CA 的 ERwin（www. ca. com/us/datamodeling. aspx）、Poseidon for UML（www. gentleware. com）和 SmartDraw（www. smartdraw. com）。

文献注解

E-R 数据模型由 Chen[1976]提出。使用扩展 E-R 模型的关系数据库的逻辑设计方法学由 Teorey 等[1986]给出。由美国国家标准与技术研究院（NIST）所发布的信息建模综合定义（IDEF1X）标准 IDEF1X[1993]定义了 E-R 图标准。然而，目前使用的 E-R 表示法有很多种。

Thalheim[2000]提供了关于 E-R 建模研究的一本详尽的教科书。Batini 等[1992]以及 ElMasri 和 Navathe[2006]给出了与此相关的基本教材。Davis 等[1983]提供了关于 E-R 模型的论文集。

321
|
322

到 2009 年为止，UML 的版本已经升至 2.2，而 UML 2.3 版本也接近最终版。关于 UML 标准及工具的更多信息，请参看 www. uml. org。