

第3章 数据库设计

本章目标

数据库系统设计技术是建立数据库及其应用系统的技术。它是信息系统建设和开发中的核心技术，特别是对于复杂的实际应用系统，数据库系统的设计显得尤为重要。数据库是数据库系统的一个重要组成部分，数据库设计是指针对一个给定的应用环境，构造最优的数据库模式并建立数据库，使之能够有效地存储数据，并满足各种具体应用需求的过程。数据库是数据库系统的数据来源和基础。因此，数据库设计也是数据库系统设计中的核心和基础。本章将简要介绍数据库系统设计整个过程，重点论述数据库设计的主要内容。通过本章的学习将达到如下的目标：

- 了解数据库系统规划与设计过程，并理解数据库系统生存周期的主要阶段；
- 掌握数据库设计的主要阶段，并重点掌握数据库的概念设计与逻辑设计；
- 掌握数据库概念设计的工程化方法和步骤，并能够熟练使用实体—联系（ER）模型，按照工程化方法和步骤要求，针对具体应用进行数据库概念设计；
- 掌握数据库逻辑设计工程化方法和步骤，并能够按照工程化方法和步骤要求，进行数据库逻辑设计，从概念模型得到 RDBMS 支持的逻辑模型；
- 理解和掌握关系数据库设计的基本要求和规范化理论，并能够对数据库设计过程中遇到的问题进行分析和解决，能够对一个完成的数据库逻辑设计进行检查和评价。

3.1 数据库系统设计概述



内容提要

数据库系统是一个复杂的计算机软件系统，因此，在数据库系统实现之前应进行充分的设计。本节简要论述数据库系统设计的主要阶段和过程，重点是其中的应用程序设计和数据库设计两个部分。

本节主要包含如下内容：

- 数据库系统的生存周期；
- 数据库应用程序设计；
- 数据库设计；
- 数据库三级模式结构。

3.1.1 数据库系统的生存周期

数据库系统通常是一个庞大且复杂的软件系统，因此，与其它软件系统一样，在实现和部署之前，应充分地进行规划、分析和设计，以保证它最终能够满足用户的使用要求。为了更好地理解软件系统生命周期的重要性，我们先简单回顾一下软件发展的历程。

计算机技术的快速发展，特别是网络的普及和发展，使得软件早已超过硬件而成为计算机系统成功的关键因素。但是在过去的几十年，软件的发展相对于硬件却显得暗淡无光，至少没有硬件发展那样令人瞩目，但人们对软件应用的需求一直在高速增长。特别是进入网络时代以来，人们的工作和生活更是离不开软件。随着软件应用的激增，从小型的、相对简单的只有几十行代码的应用程序，到大型的、复杂的具有几十万行代码的应用程序；从单机环境下的应用程序到网络环境下的应用程序，这些程序都需要持续的维护，包括修正已经发现的错误，实现用户新的需求，将软件移植到新的平台或者升级之后的平台上。自从软件产生以来，软件项目开发期限的延期和经费的超支经常发生，软件质量不可靠，维护困难并且性能达不到预期要求也一直存在，即使今天，软件的可靠性和安全等问题仍然有待于解决。由于这些问题的存在，导致了众所周知的“软件危机”（software crisis）的出现。尽管“软件危机”一词是在 20 世纪 60 年代第一次被提出，但时至今日，危机依然存在。

20 世纪 90 年代中期，不同组织和机构曾经对软件工程产业的现状进行了至少 3 次重要的调查和分析。三次调查和分析都得到了相同的、具有普遍性的结论：软件的成功率非常低。下面列出其中的一些调查和分析结果，用以说明软件危机的存在：

- 1) 80%~90%的项目没有达到预定的性能目标。
- 2) 大约 80%的项目开发延期或超预算。
- 3) 40%以上的项目失败或被放弃。
- 4) 近 40%的项目需要技能培训。
- 5) 不到 25%的项目能够使应用和开发技术目标一体化。
- 6) 只有 10%~20%的项目能完全达到成功标准。

导致项目失败的主要原因如下：

- 1) 缺少完整的需求定义。
- 2) 缺少先进的开发方法学。
- 3) 无法将设计分解成易于管理的部分。

为了解决这些问题，软件工程领域的专家、学者以及工程师们一直在不懈努力，从软件开发技术、软件工程方法和过程以及软件项目组织与管理等诸多方面进行研究，取得了一系列成果，从而能够在一定程度和范围内控制软件危机，否则，我们今天的生活可能不会象现在这样，与软件联系得如此紧密。为解决这些问题而提出的一种方案是采用结构化的软件开发方法，称之为软件开发生命（生存）周期（Software Development Lifecycle, SDLC）。

概括而言，这种结构化的思想就是把软件开发分为几个不同的阶段，每一个阶段完成系统不同的任务，最后形成一个完整的系统。基于这种思想产生了不同的软件开发过程模型，针对不同软件类型和特点，可以采用不同的过程模型，这些内容在软件工程相关书籍和论文中有详细论述。由于本书所

论述的数据库系统开发过程涉及到了软件开发生命周期，在此仅仅提及，但不作详细论述，有兴趣的读者可以查阅相关书籍和论文。

由于数据库系统是庞大且复杂的软件系统，因此，数据库系统的开发同样应该遵循软件系统开发生命周期。

数据库系统设计的目标，除了应满足最终应用系统的数据需求之外，还应能够支持最终应用系统的操作和业务。例如，对于图书馆数据库系统的设计目标，除了应该能够满足图书馆中图书信息存储之外，还应该能够支持图书馆的各种基本操作，如查询图书、借书与还书等。因此，数据库系统设计至少应该包含两个方面内容或阶段：一方面是满足应用系统数据存储需求的数据库的设计；另一方面满足应用系统业务操作或行为需求的应用程序的设计。除此之外，数据库系统设计还有其它阶段和内容，本书将主要涉及数据库设计与应用程序设计这两个方面的内容。在进一步论述数据库设计和应用程序设计这两个方面内容之前，有必要先从软件工程的角度，浏览一下数据库系统的软件开发生命周期或者叫软件生命周期，从而对数据库系统的设计与开发有一个整体和概念上的认识。数据库系统整个生命周期如图 3.1 所示。

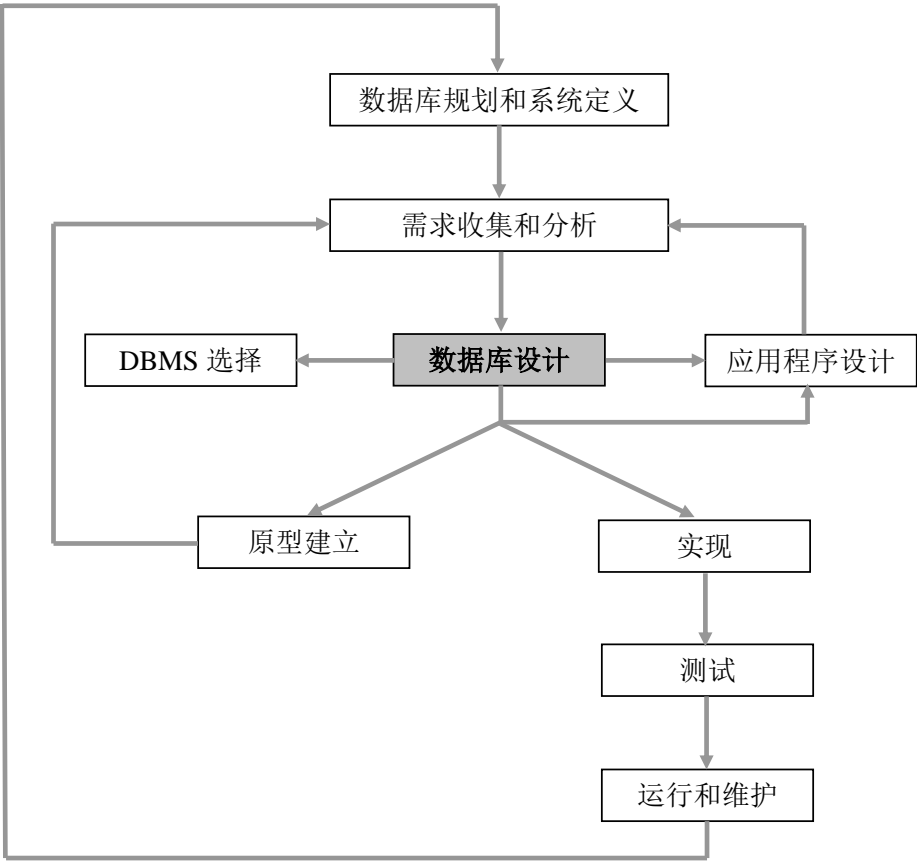


图 3.1 数据库系统生命周期

在数据库系统生命周期各个阶段中，数据库规划是一种管理活动，其目的是使数据库以及基于数据库的应用程序最终能够高效和有效地实现。该阶段的主要活动包括：定义项目的任务描述、确定项目目标和实现路线，甚至还包括相关标准的选择和建立，例如，数据收集整理格式、需要文档类型以

及格式等。系统定义是描述数据库以及应用程序的范围边界及主要的用户视图。用户视图是从一个单独的工作角色（如教师或者学生），来定义数据库应用程序的需求，并加以描述。一个数据库系统可能会有多个用户视图，确定用户视图是开发数据库系统的一个重要方面，同时，用户视图可以帮助我们确定在系统需求分析时，没有遗漏主要的数据库系统用户。确定系统范围边界非常重要，不然在后续阶段，可能会由于用户需求无限制增长而使系统陷入不可控状态。在建立边界时，不应该只考虑用户当前应用，还应该适当考虑用户后续可能出现的需求和应用领域的发展状况。

需求收集和分析阶段是在数据库规划和系统定义范围内，收集和分析系统内将由数据库支持的那部分信息以及将由应用程序支持的事务。对需求的收集和分析最终要形成文档，通常成为数据库应用的需求说明书。对于应用程序，可能还需要单独形成文档，通常称为软件需求说明书。在需求收集和分析过程中，有多种技术和方法可以运用。

实况发现（Fact-Finding Technique）是一种最为常用的需求收集技术。它是使用面谈和调查表等技术手段收集关于系统、需求以及用户倾向等实况的正规过程。在软件工程的其它阶段，也经常使用该技术进行实况发现。分析文档资料、面谈、观察工作流程、问卷调查等手段都属于实况发现技术。这些实况发现技术具有各自的优缺点，在不同情况下应选择不同的技术。

对于系统中可能存在的多个用户视图，在需求分析过程中也有不同的处理方法，通常有三种主要的方法：集中式方法、视图集成方法以及两种方法的混合。

集中式方法是系统中存在的每个用户视图的需求合并成单一的需求集合，针对这一整体需求进行后续的数据库设计工作。当各个用户视图的需求存在明显重叠，并且数据库系统不是很复杂时，可以考虑采用这种方法。

视图集成方法是针对每个用户视图的需求，分别进行后续的数据库设计工作，在数据库设计的过程中再整合针对每个用户视图建立的数据模型。通常，当各用户视图之间存在明显差别，而且整个数据库系统比较复杂时，可以考虑采用这种方法，因为该方法能够将工作分解为易于管理的部分。

针对比较庞大的系统，在需求分析过程中，可以考虑采用两种方法的结合，在需求的不同部分或者不同阶段，分别采用集中式方法或者视图集成的方法。

对于需求收集和分析的详细过程以及所采用的技术和方法，在软件工程或者需求工程相关书籍中均有详细描述，本书在此仅作简单介绍，以使读者对数据库系统整个过程有所认识 and 了解。

数据库设计阶段是根据系统需求创建支持企业运作或者企业目标的数据库的过程，这也是本书的重点内容，本章后面将会详细论述。从图 3.1 可以看出，应用程序设计阶段和数据库设计阶段是可以并行活动的。一般情况下，不可能在数据库设计完成之前就完成应用程序设计，因为应用程序的设计需要数据库的支持，应用程序设计和数据库设计之间必然存在信息交流。

在以上阶段完成之后，根据项目的不同可能采取不同实现方案。如果项目需求分析比较充分，用户对系统功能定义完整并且准确，系统操作以及界面风格确定，那么，接下来可能会直接进入系统实现阶段，否则可能会采用原型法。原型是针对最终系统的一个简略的工作模型，通常不会拥有最终系统所要求的全部特性和功能。建立原型的主要目的是，允许用户使用原型，以确定系统中哪些功能比较充分，哪些功能尚不完善，从而进一步提出更准确的功能定义。通过这种方式可以很大程度上帮助

用户和系统开发人员更准确了解用户的需求。通过原型进一步确定系统功能之后,再进入系统实现阶段。系统实现阶段之后是测试和系统运行维护阶段。这些阶段的内容描述在软件工程相关书籍中有详细介绍,下面仅针对本书涉及到的数据库系统生命周期中两个主要阶段作进一步论述。

3.1.2 应用程序设计

本书所提及的应用程序主要是指数据库应用程序。应用程序设计除了要满足用户需求中的各种功能和业务之外,还应该负责提供恰当的用户界面。因此,应用程序设计主要包含两方面内容:事务设计和用户界面设计。前者主要关注系统功能实现,而后者则主要关注向用户提供友好的使用方式。

1) 事务设计

在讨论事务设计之前,先说明事务的含义,本书在第四章事务管理内容中还将继续讨论事务,在此,仅从用户角度对事务进行描述。

事务(Transaction)是对数据库进行访问或修改的一个或多个操作,这组操作组成一个单位,共同完成一个任务。

事务是现实世界中事件的表示,例如,银行转帐是一个事务,它包括从一个账户扣除转帐金额,在另外一个账户上增加转帐金额,这两个操作组成一个单位,共同完成转帐业务。登记学生成绩、增加一门课程、借阅图书等,都是事务。在应用程序设计中,我们从用户业务操作中抽象出主要活动以及它们之间的联系,组成一个个逻辑单位,对每个逻辑单位,指定其输入数据、对数据的操作和输出数据,应用程序的功能就是由这些逻辑单位的活动组成。其中每个逻辑单位对数据的请求对应一个对数据库的操作序列,即事务。

一个事务可能由多个对数据库的操作组成,由于这多个操作共同完成一个任务,因此,应该保证同一个事务中的多个操作具有一些特性。例如,对于银行转帐事务,至少包含两个主要数据库操作,一个操作是更新贷方账户,扣除贷出金额;另一个操作是更新借方账户,增加贷入金额。如果资金已经从贷方账户贷出,但尚未划入借方账户,也就是说两个操作只完成了第一个,就会出现问题。因此,数据库管理系统应该保证同一个事务中的多个操作要么都做,要么都不做。除此之外,事务还应该具有其它方面的特性,在本书后面将详细讨论。

事务设计的目的在于根据用户需求确定数据库所需要的事务,并用文档记录这些事务的功能和特性。事务设计主要关注以下几个方面:

- (1) 事务中将要操作的数据;
- (2) 事务对数据处理的功能特性;
- (3) 事务的输出;
- (4) 事务对用户的重要性;
- (5) 事务的使用频率。

2) 用户界面设计

用户界面设计是最终用户和系统进行交互的窗口,因此,用户界面设计主要关注数据和用户操作的展现形式,它应该遵循软件工程中用户界面设计的基本原则。在数据库应用程序中,表单或者报表应用一般较多,它们的布局和外观应该遵循一些基本原则,例如,表单或者报表中的控件布局应该均

衡、规则，外观应该一致，标题表达的信息应该明确和清楚；表单或者报表中的控件排列顺序应该符合逻辑，而且应该具有一定的异常处理和自检查能力。除此之外，对表单和报表的设计还有许多细节的要求。对于更多关于用户界面设计的内容，在人机界面或者以用户为中心的软件设计相关书籍中有详细论述，本书不再赘述。

3.1.3 数据库设计

前面已经提到，数据库设计阶段的主要活动是创建支持企业运作或者企业目标的数据库，具体而言就是根据用户或企业对数据的需求，针对特定的数据模型建立数据库模式的活动。数据库设计过程其实也就是建模过程，经历了由现实世界到信息世界，再到计算机世界。在这个过程中的不同阶段，模型针对的使用对象也不尽相同，因此，应该用不同类型的数据模型对建模结果进行描述，从而满足不同阶段模型的使用需要。

一般情况下，数据库设计包含三个主要阶段，它们分别是：概念设计、逻辑设计和物理设计。这三个设计阶段中的每个阶段将产生不同的结果模型，具有不同的用途和目的。

1) 概念设计

概念设计是根据企业的目标，针对所设计系统中的数据需求建立模型的过程，它是数据库设计的第一个阶段，也是整个数据库设计的关键。

在此阶段，通过对用户需求进行综合、归纳与抽象，形成一个独立于所有物理因素的模型，并且完全独立于实现细节，例如，数据库系统在实现中将采用的 DBMS 软件、应用程序结构、编程语言、硬件平台或其它任何实现上的考虑。

概念设计的目的是充分挖掘系统对数据的需求，并进行归纳和抽象，在此基础上所建立的模型应该容易为设计人员和用户所理解，从而为设计人员和用户之间提供一个交流的桥梁，同时，该模型应该能够较准确表达用户需求，减少二义性。目前常用的模型是实体—联系（Entity-Relationship，ER）模型，本书后面将详细介绍这种模型。

概念设计的结果将得到概念数据模型，通常为 ER 模型，它将是数据库设计下一阶段，即逻辑设计的信息来源。因此，在概念数据模型建立过程中，模型应该被测试并验证是否满足用户的需求。

2) 逻辑设计

逻辑设计是将概念设计阶段所得到的概念数据模型转化为目标 DBMS 所支持的数据模型的过程。

逻辑设计仍然与所有的物理因素无关，但它是针对特定数据模型进行设计的，建立在目标 DBMS 所支持的数据模型的基础之上。也就是说，逻辑设计是在知道目标 DBMS 所支持的基本数据模型的条件下进行设计的。因此，在建立逻辑模型之前，应该先确定最终的目标 DBMS 所支持的数据模型，例如，关系模型、层次模型、网状模型或者面向对象模型等，针对目标 DBMS 所支持的模型不同，概念模型转化所得到的逻辑模型结果也不相同。当前广泛应用的 DBMS 大部分都支持关系模型，因此，本书后面只重点讨论如何把概念模型转化为关系模型。

逻辑设计的目的是根据目标 DBMS 所支持的数据模型的特点，把概念数据模型所表达的信息转化为用目标 DBMS 所支持的数据模型进行表达和描述。在逻辑设计阶段得到的数据模型应该容易为设计人员和支持该种数据模型的 DBMS 所理解，从而成为设计人员和某种 DBMS 之间的桥梁，另外，由于该

模型能够为 DBMS 所理解和使用，因此它应该能够准确表达设计人员的意图，并且无二义性。

逻辑设计的结果将得到逻辑数据模型，由于目前广泛应用的 DBMS 大部分为关系 DBMS，支持关系模型，因此该阶段设计的结果通常为关系数据模型，它将是数据库设计下一阶段—物理设计的信息来源。在逻辑数据模型建立过程中，模型应该被测试并验证满足用户的需求，同时，它还应该满足它针对的特定的数据模型（关系模型）的要求和规范。本书后面将详细介绍针对关系模型的规范化要求。

3) 物理设计

物理设计是对数据库在辅助存储上的实现进行描述的过程。包括描述数据库最终的文件组织、高效数据访问的方法，以及所有完整性约束和安全措施等物理实现细节。

物理设计是数据库设计的最后阶段。在这一阶段，设计人员要确定如何面向具体 DBMS 最终实现该数据库。因此，在进行物理设计时，必须首先确定最终将要使用的具体 DBMS，例如，Oracle、PostgreSQL 或者 SQLServer 等。从图 3.1 所展示的数据库系统的生命周期中也可以看到，在数据库设计阶段有一个 DBMS 选择阶段，它为物理设计提供了基础和支持。

当前流行的 DBMS 大多采用固定的物理设计，它们能够自动选择高效的访问路径，能够自动优化性能，从而减少手工对性能调优的操作，而且这种趋势越来越明显。并且，DBMS 也正向智能化方向发展，以方便用户操作和使用，减少用户对数据库的物理干涉。此外，由于物理设计跟具体 DBMS 密切相关，需要对具体 DBMS 有深入了解，不能一概而论。因此，本书仅着重对数据库设计的前两个阶段，即概念设计和逻辑设计进行讨论，这两个阶段在数据库设计中具有一定通用性。

在数据设计过程中，概念设计和逻辑设计对一个系统的成功具有决定性影响，因为这两个设计阶段的结果代表了用户需求，它们的设计结果如果不准确或者与用户需求有偏差，将导致最终系统不能满足用户或企业的使用要求。数据库设计是一个反复修改和不断完善的过程，好的数据库设计应该能够根据需求变化作出相应调整，这也是一个好的数据库设计的标志。因此，在数据库设计阶段花费必要的时间和精力，从而得到尽可能好的数据库设计是值得的。

3.1.4 数据库三级模式结构

根据上述数据库设计过程，可将数据库设计过程分为三个阶段，其中每个阶段都有不同的目的和结果，从而能够更好地把用户数据需求转化为数据库的物理实现。其实，早在 1975 年，美国国家标准协会（ANSI）的标准规划和需求委员会（SPARC）就提出了一个数据库三级体系结构，该体系结构虽然没有成为标准，但却被普遍接受和认可，各种数据库厂商也对这三级体系结构提供不同程度支持。事实上，数据库设的概念设计、逻辑设计和物理设计三个阶段与 ANSI-SPARC 提出的数据库三级体系结构是相对应的。

根据 ANSI-SPARC 的三级体系结构，可将数据库中的数据抽象描述为三个不同的层次或者模式，分别称为外模式、概念模式和内模式，如图 3.2 所示。当然，数据最终仅存储于物理文件之中，不同模式只不过是不同层次对数据进行了不同的描述。

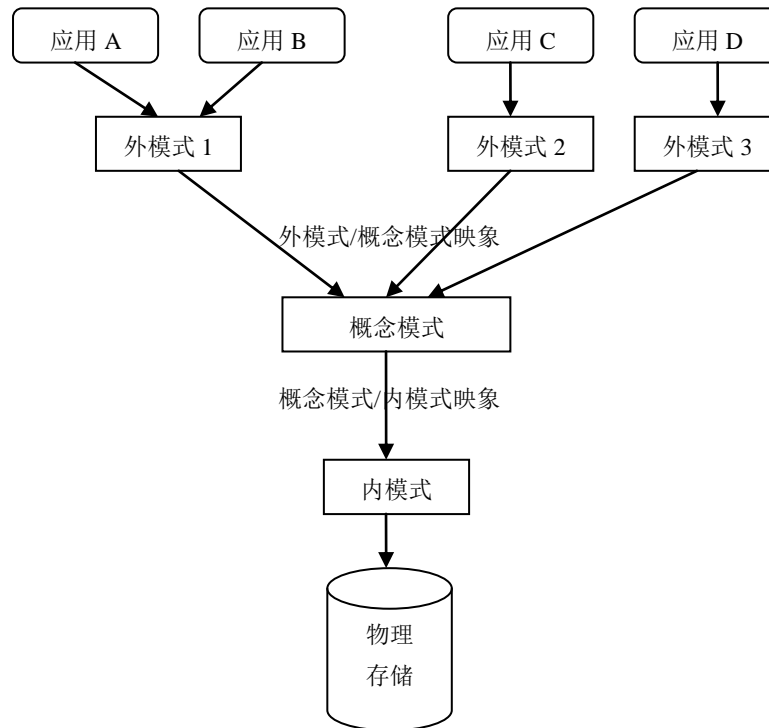


图 3.2 ANSI-SPARC 三级模式体系结构

1) 外模式

外模式是数据库的用户视图。外模式的每个部分描述了与用户相关的数据库部分。在外模式中，每个用户都用其熟悉的方式表示“现实世界”，不同用户对现实世界的关注视角或侧面不同，导致每个用户对“现实世界”的描述也是不同的，因而，在数据库设计的第一个阶段—概念设计中，就可能会产生针对不同用户的多个概念数据模型，但这些不同的概念数据模型经过数据库设计的其它阶段，最终将进入到一个数据库之中。同样道理，不同用户对同一个数据库中数据的理解和使用需求也是不同的，因此，从同一个数据库中取出的数据，最终展现给用户的表现形式也可能不同。

例如，一些用户对“日期”的描述可能按日、月、年的形式，而另一些用户的描述形式可能是年、月、日的形式，这两种对日期的不同描述属于外模式范畴，它们在最终实现的数据库中其实对应的是同一个数据。同理，不同用户从数据库取出同样的日期数据，根据要求不同，展现形式也不一样，一些用户以日、月、年的形式展现，而另一些用户则可能以年、月、日的形式展现。

另外，一些用户视图所需要的数据也可能在数据库并不存在，它是根据从数据库取出的数据经过计算而得到的。不同用户从数据库取出同样的数据，经过不同计算，可能会得到不同的结果，以不同的形式展现。例如，在学生信息数据库系统中，可能需要查看学生年龄，而学生年龄在数据库中并不实际存在，但是它可以通过学生出生日期经计算得到，而学生出生日期在数据库中是存在的。

2) 概念模式

概念模式是数据库的整体视图，它描述了哪些数据存储于数据库之中以及这些数据之间的联系，这一层包含了整个数据库的逻辑结构。概念模式处于三级模式结构的中间层，地位尤为重要。它对上层的外模式，要支持所有用户视图，即所有用户可访问的数据都能通过概念模式导出；对下层的内模

式，要提供数据库最终将要存储的所有数据信息，但不包含任何与存储相关的详细信息。

例如，在学生信息数据库系统中，对于用户教师，可能需要选修某一门课程学生的信息，而学生可能需要某一门课程的任课教师信息，它们分别属于两个不同的用户视图，或者外模式。在概念模式中，需要对这两个不同的外模式进行集成，以得到一个统一的概念模式，它要包含所有不同用户视图需要的数据信息。对于本例，概念模式可能包含教师信息、学生信息、课程信息、教师任课信息以及学生选课信息这几个关系，同时还要给出这些关系之间或者关系内部的数据联系。从而使得该概念模式能够支持所有外模式或用户视图。另外，由于概念模式不能包括任何依赖于存储的详细信息，因此，对于其中的每个关系的描述，概念模式只能描述关系中属性的数据类型（例如，整型、实型或字符型），以及它们的长度（例如，数字或字符的个数），但不能指定与具体存储相关的信息（例如，数据所占用的字节数）。

3) 内模式

内模式是数据库在计算机上的物理表示，它描述数据最终是如何存储在数据库中的。内模式包括在存储设备上存储数据所使用的数据结构和文件组织，主要涉及的内容包括：数据和索引的存储空间分配，存储记录的描述（记录各个数据项存储空间大小），以及数据压缩和加密技术等。

内模式之下是具体的物理存储，它可能在 DBMS 的指导下受操作系统的控制，然而，DBMS 和操作系统在物理存储上的功能分割并不是十分清晰，并且因系统而异。

前面曾经提到，ANSI-SPARC 的三级模式体系结构描述了数据库中相同的数据在不同层次上不同的表现方式。因此，DBMS 就应负责提供数据在这三级模式之间的映射。具体地说，DBMS 要提供外模式和概念模式之间的映射，从而保证不同用户视图所需要的数据能够从概念模式中找到，通过概念模式与内模式之间的映射，以保证概念模式中的数据能最终从物理存储中找到，然后把数据返回给不同用户。

ANSI-SPARC 提出三级模式体系结构的一个主要目的是保证数据的独立性，这意味着，通过三级模式体系结构以及它们之间的映射，能保证对较低层模式的修改不会对较高层模式产生影响，这也是我们前面在数据库特点中曾经提到的数据独立性。ANSI-SPARC 通过三级模式结构以及它们之间的映射提供了如下两种数据独立性。

(1) 逻辑数据独立性

逻辑数据独立性是指外模式或者应用程序不受概念模式变化的影响。这意味着我们可以通过对概念模式进行改变、扩充或者进一步完善，以达到对数据库扩充或者改变的目的，而这些改变不会影响与这些改变无关的外模式或者外部应用程序。

例如，在教学信息数据库系统中，假设在概念模式下有学生关系，其关系模式为 Student(studentNo, name, idCardNo, DOB, enrollmentYear)，如果现在我们改变该关系模式，在关系模式中增加一列属性“籍贯”，此时该关系模式将变为 Student(studentNo, name, idCardNo, DOB, enrollmentYear, nativePlace)，这个改变不会影响当前使用该关系的应用程序，应用程序可以按照原来的方式正常使用此关系模式。不过，如果现在需要应用程序使用新增加的属性，可能就需要对应用程序加以改变。严格地讲，这种改变应该是需求变化导致的。对于那些没有使用该关系模式的应用

程序来讲，更是没有任何影响。当然，如果对 Student 模式的改变是删除其中的一个不再使用的属性，那么原来使用到该属性的应用程序就必须加以改变，但这种改变仅限于使用了该属性的那些应用程序，对其它应用程序不会带来影响。

（2）物理数据独立性

物理数据独立性是指概念模式不受内模式变化的影响。即对内模式的修改（例如使用不同的文件组织方式或存储结构；使用不同的存储设备；文件路径发生变化；修改索引等。）不会影响概念模式和外模式，也不影响应用程序的执行结果，唯一能体会到的可能是性能上的变化，而这正是改变内模式的常见原因。

在设计数据库时，如果完全按照 ANSI-SPARC 三级模式体系结构，经过两次映射，设计效率将下降，且最终对数据库中数据访问的效率也会降低。但是，这种结构能够提供更强的数据独立性，为数据库的实施和维护带来了方便。

上述数据库设计的三个阶段与 ANSI-SPARC 三级模式体系结构是相对应的，其对应关系如图 3.3 所示。

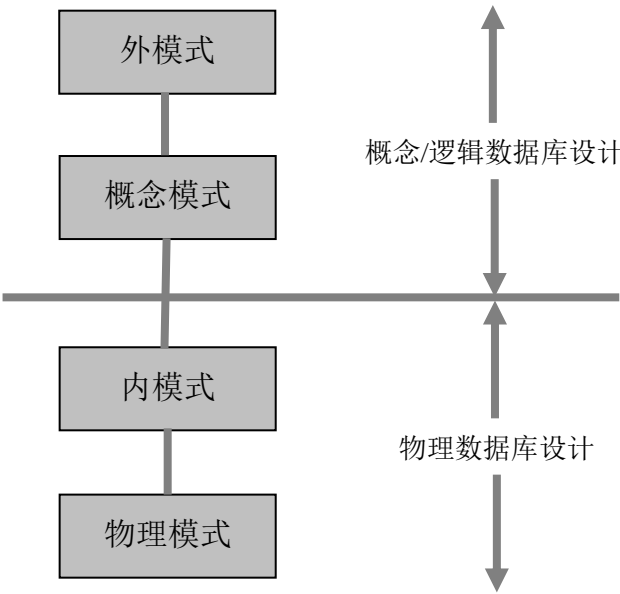


图 3.3 数据库设计与 ANSI-SPARC 体系结构对应关系

3.2 概念数据库设计



内容提要

3.1 节对数据库系统生命周期进行了简单的介绍，数据库设计是数据库系统生命周期中的重要阶段。为了使数据库最终能够更准确地反映现实世界，我们把数据库设计分为三个主要阶段，它们分别是：概念设计、逻辑设计和物理设计。这三个设计阶段中的每个阶段将产生不同的结果模型，具有不同的用途和目的。概念设计是数据库设计的第一个阶段，该阶段将需求分析阶段所得到的用户需求转化为概念模型。该模型是用户与数据库设计人员交流的桥梁，目前常用的概念模型是实体—联系（ER）模型。本节将详细讨论数据库概念设计的方法和过程，主要包含如下内容：

- 实体—联系（ER）建模技术；
- 概念数据库设计过程。

3.2.1 实体—联系（ER）建模技术

数据库系统生存周期的需求收集和分析阶段一旦完成，数据库应用的需求文档便产生了，因此可以进入数据库设计阶段。

数据库是由数据库设计人员根据最终用户的需求而设计，编程人员编写与数据库交互的应用程序来满足最终用户使用要求。由于设计人员、编程人员和最终用户分别以不同的方式和角度来看待数据以及数据库的使用，这也成为数据库设计最困难的一个方面。然而，要保证数据库设计能够最终满足用户的需求，又必须要求不同用户对数据库中的数据需求有一个共同的认识。

为了使不同用户对数据及其在企业中的使用有一个准确的、一致的认识和理解，不同用户之间就需要一种模型来进行交流，并且该模型应该是非技术和无二义性的。非技术的目的是使不同用户都能够理解模型，并且能够通过模型进行交流；无二义性的目的是使不同用户对模型的认识和理解是准确的和一致的。用自然语言描述显然是不符合这种模型要求的，因为自然语言表达不够准确，而且容易产生二义性。实体—联系（Entity-Relationship, ER）模型能满足这一要求型，并且也是目前最为常用的数据库概念设计阶段模型，它为数据库概念设计提供了一种半形式化的标注方法。

ER 模型是一种自上而下的数据库设计方法，该方法从确定一些重要的数据和这些数据之间的联系开始，然后逐步增加更多的细节信息。ER 模型是所有数据库设计人员都应该掌握的一种重要技术，是数据库设计的基础。

尽管 ER 模型已经得到广泛应用和较一致的认识，但是对 ER 模型的表示方法却不止一种，例如，常用的标注方法有 Crow's Feet 标注方法、Chen 标注方法以及 UML(Unified Modeling Language, UML) 标注方法。这些标注方法功能基本一致，但标注之间尚存在着一些差异。其中，统一建模语言 UML 是 20 世纪 80 年代到 90 年代出现的面向对象分析和设计技术的后继产物，对象管理组织（Object Management Group, OMG）现在已经将 UML 标准化，使其成为了标准的建模语言。本书也将使用 UML 这种日益流行的面向对象建模语言的图形化表示方法，但在建模过程中，仍使用数据库术语对模型进行描述。

ER 模型一般使用图形化的方法表示，但是仅仅依靠图形化的方法是不够的，仍然需要文档进行描述。这些文档一般是针对 ER 模型中出现的各种元素进行描述，后面还将提到每个阶段所需要的文档。

下面首先对 ER 模型图形化建模过程中常用的基本建模元素逐一进行描述。

1) 实体与实体型

实体（Entity）是现实世界独立存在并可以唯一标识的对象。**实体型（Entity Type）**是一组具有相同属性的对象。在建模过程中，如果要表示每个实体以及实体之间的联系，那么模型将会由于企业中实体的众多并且联系细节的复杂而变得难以表述和理解。因此，ER 模型建模过程中，通常仅对实体型以及实体型之间的联系进行标识，而不对每个实体以及实体之间的联系进行标识。对于实体和实体型，在不引起歧义的情况下，本书通常都统一使用“实体”一词表示。

一个实体的独立存在，既可以是物理上存在的对象，也可以是概念上存在的对象。例如，学生和教师都是物理上存在的对象，而学生选课情况或者银行账户，并不是物理上存在的实体，它们属于概

念上存在的实体。

不同实体型可以通过一个唯一的实体名和一组属性来区分，一个数据库通常包括很多实体型。在 ER 模型的图形化中，实体型用矩形表示，矩形里面标识该实体的名称，实体名通常为名词，英文一般用单数形式，实体名的每个单词第一个字母大写。图 3.4 显示了实体型 Student 和 Class 的图形化表示。



图 3.4 Student 和 Class 实体型的图形化表示

2) 联系与联系型

联系 (Relationship) 是两个或更多实体间有意义的关联。**联系型 (Relationship)** 是实体型间一组有意义的关联，或者说联系型是一组相似的联系。前面在实体与实体型的定义中已经说明，在 ER 模型中，一般针对实体型标识，而不会针对具体实体个体进行标识。同理，ER 模型一般针对联系型标识，而不会针对具体联系进行标识。对于联系和联系型，在不引起歧义的情况下，本书通常都使用“联系”一词表示。

不同联系可以通过一个唯一的联系名来区分，有时联系也可能会有属性，一个数据库通常包括很多联系。在 ER 模型的图形化表示中，对于两个实体型间的联系，通常用连接两个实体型的直线表示，并在直线上方标识该联系的名称，联系名通常为动词或动词短语，与实体型一样，联系名每个单词的第一个字母大写。一个联系只应标记为一个方向，因此，在联系名确定之后，一般会在联系名旁边放置一个箭头符号来表示联系的方向。例如，图 3.5 标识了学生实体 Student 与课程实体 Course 间的选课联系 Elects，它表示学生可以选修课程。

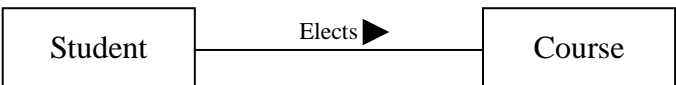


图 3.5 联系的图形化表示

两个实体间也可能会发生多于一种联系，在这种情况下，可以使用角色名来区分实体每次参与联系的角色。例如，图 3.6 标识了学生 Student 和班级 Class 两个实体间的两种联系。具体说，Student 和 Class 两个实体间的一个联系是“学生管理班级”，在这个联系中，学生 Student 的角色为班长 Monitor，它管理一个具体的班级 Class。类似地，在联系“班级拥有学生”中，学生 Student 的角色为班级成员 Member。

当参与联系的实体在联系中的功能无二义性的情况下，通常不需要使用角色名。例如，对于两个实体间只有一种联系的情况下，参与联系的双方功能一般不会产生二义性，因此一般不需要标识角色名。

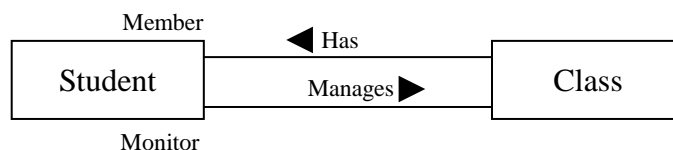


图 3.6 两个实体间的两种联系

联系型的度 (Degree) 是指参与某一联系型的实体型的数目。包含在一个特定联系型中的实体型被看做该联系型的参与者，一个联系型参与者的数目称为该联系型的度。度为 2 的联系称为二元联系，图 3.5 表示的实体型 Student 和 Course 之间的 Elects 联系就是一个二元联系。图 3.6 中的两个联系也均为二元联系。一般情况下，ER 模型中的大多数联系都是二元联系。

度为 3 的联系称为三元联系，度大于 3 的联系称为多元联系。某些情况下，在 ER 建模过程中也可能会遇到三元联系或者三元以上的多元联系。在 ER 模型的图形化表示中，一般用菱形表示度大于 2 的联系，在菱形中标识联系名，在这种情况下，通常省略指示联系方向的箭头。例如，对于培训公司，某一支机构具有若干咨询师，客户通过咨询师推荐注册到某个分支机构，从而参加某种培训，图 3.7 标识了客户、咨询师以及分支机构三个实体间的一个三元联系。三元联系表示的意义并不等价于三个实体间的两两二元联系。对于图 3.7 标识的三元联系，它表示了“一名客户通过某一咨询师注册到了某一支机构”。如果把图 3.7 的三元联系修改为三个实体间的二元联系，将会产生三个二元联系，但是这三个二元联系与三元联系的含意不同，并且在一般情况下，它们并不等价。

对于本例的三元联系，若将其修改为三个实体之间的两两二元联系，则咨询师、分支机构以及客户之间的三个二元联系分别为：分支机构与咨询师之间的联系 Has<Branch, Advisor>；分支机构与客户之间的联系 Registers<Branch, Client>；咨询师与客户之间的联系 Serves<Advisor, Client>。它们表示了分支机构拥有咨询师，客户注册到分支机构，咨询师为客户服务。但是它们不能表示某一客户是通过某一具体咨询师注册到某个分支机构这个含意，也就是说，在某些情况下，通过这三个两两二元联系，不能确定“某一客户是通过哪位咨询师注册到某一支机构”这一事实，而这三个实体间的三元联系却能够表达这一事实。但根据联系的约束不同，某些情况下三个实体间三元联系也能够用它们之间的两两二元联系代替。在后面讨论了联系约束的内容之后，可以继续思考并讨论这个案例。

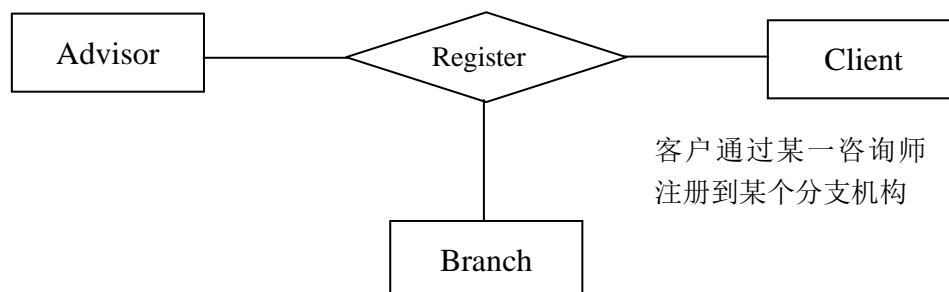


图 3.7 三元联系的表示

在本课程中的 Teaching 案例中，如果多个系 Department 可以设置同一门课程 Course，且同一门课程也可能有多个教师 Teacher 讲授，但是某个教师只能为某一个具体系讲授某门课程，那么，课

程、系与教师之间也将是一个三元联系。

ER 建模过程中，一般很少会用到三元或者三元以上的多元联系，因此不再详细讨论。多元联系所表达的含义可以根据三元联系类推。

联系除了可以在两个或者多个不同实体间发生外，也可以在同一个实体之上发生，称为递归联系。

递归联系 (Recursive Relationship) 是同一个实体型参与次数大于 1 的联系型，同一实体型每次参与的联系具有不同的角色。

考虑教学信息数据库系统中的课程实体 Course。假设对某一课程的学习可能需要具备一定的基础知识，也就是说课程可能具有先修课，而先修课也是一门课程。这时课程实体就与自身发生了联系。该课程实体两次参与联系所扮演的角色不同：一次参与的角色为后续课程；另一次参与的角色为先修课。在递归联系中，可以使用角色名来区别实体每次参与联系的角色。例如，图 3.8 标识了课程实体的递归联系。

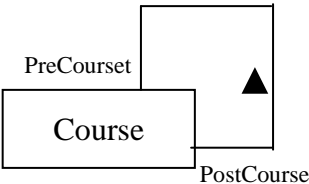


图 3.8 递归联系

3) 属性

属性 (Attribute) 是实体型或联系型所具有的某一特性。例如，学生实体具有属性 studentNo, name, sex 等。

有时联系也可能具有属性，例如，在教学信息数据库系统中，学生实体 Student 与课程实体 Course 之间有联系选课 Elects，该联系应该具有属性成绩 score 和得分日期属性 reportDate，它们表示学生在选修了一门课程，在某时得到一个该门课程的成绩。

每个属性一般都与一个取值集合相关联，这个集合称为**域 (Domain)**，本书前面在关系模型中曾经给出过域的定义，它与这里的定义是一致的。例如，学生实体具有性别属性 sex，其取值范围只能是“男”或者“女”，则可以为 sex 属性定义一个域{男，女}；对年龄属性 age，其取值范围应该固定为两位数字，同样可以为年龄定义一个域。

多个属性可以共享一个域。例如，教师实体 Teacher 如果也有性别属性 sex，它可以与学生性别属性共享一个域。

一个完整的 ER 模型应该包括其中每个属性的域。

根据不同的分类方式，属性可以分为：简单属性和复合属性；单值属性和多值属性；导出属性等。

(1) 简单属性和复合属性

简单属性是由单个部分组成的属性，**复合属性**则是由多个部分组成的属性，并且其中每个部分都可以独立存在。例如，学生实体 Student 的学号属性 studentNo 为简单属性，它由单个部分组成。如果学生实体具有家庭地址属性 homeAddress，该属性很可能就是一个复合属性，因为它可能由某某省、

某某市、某某路某某号组成，并且组成该属性的每个部分都是可以独立存在的。某个复合属性应由哪几部分组成，用户最终的使用需求可能起决定性作用。对于 homeAddress 属性，如果用户最终要求单独使用地址中的省和市（例如，用户有查询来自某某省或某某市学生的需求，就要求单独使用省和市），那么 homeAddress 属性就由省、市和具体地址三个部分组成；如果用户最终只要求单独使用地址中的省（如果用户查询学生时，只需要根据省查询，而不需要具体到市），不要求单独使用市，那么该属性可以看作由省和详细地址两个部分组成，其中详细地址包括了市以及具体地址。

（2）单值属性和多值属性

单值属性是指对于实体型中的每个实例（即每个具体实体），该属性只有一个取值。**多值属性**则指对于实体型中的每个实例（即每个具体实体），该属性可能有多个取值。大多数的属性都是单值属性，例如，学生实体的学号属性，对于每个学生该属性只能有一个取值，因此该属性为单值属性。如果学生实体有一个联系电话属性，并且每个学生的联系电话可能有两个或者更多（比如，宿舍电话，小灵通或者手机），那么该属性就为多值属性。多值属性的取值数目有时也有限制，例如，学生电话号码属性取值数目可以限制为 1 到 3 个。

（3）导出属性

导出属性是指可以由相关的一个或者多个属性之值通过计算得到的属性。导出属性对于实体来讲并不是必须的。例如，如果学生实体具有年龄属性 age，而同时具有出生年份属性 DOB，那么 age 属性就是一个导出属性，它可以由当前日期和出生年份属性 DOB 计算得到。某些情况下，一个实体的属性也可能由其它实体的属性导出。例如，如果学生实体具有平均学分积属性，该属性就可以由学生每门课程的成绩属性以及每门课程的分计算而得到，而课程的成绩以及学分属性并不包含在学生实体中。

单个属性或者属性的组合可以构成实体的主关键字。对于超关键字、候选关键字以及主关键字，前面在关系模型中已经给出了定义，这些定义与它们在 ER 模型实体中的定义是一致的，下面对其中很重要的主关键字进行说明。

在实体中，能够唯一标识每个实体的最小属性组构成了实体的候选关键字，其中被选定用来唯一标识每个实体的候选关键字称为主关键字。主关键字可能是单个属性，也可能由多个属性组合而成，由两个或者两个以上属性组合而构成的主关键字称为**复合关键字**。

在 ER 模型的图形化表示中，如果要在一个实体中显示它的属性，可以将表示实体的矩形分为上下两个部分，上部分标识实体的名称，下部分标识实体属性。属性的命名一般为名词，如果属性由多个单词构成，除第一个单词的第一个字母小写外，其它单词的第一个字母大写。在属性的标识中，还可以针对不同的属性类别进行标识。例如，主关键字属性用 PK 标识，辅关键字用 AK 标识。图 3.9 标识了学生实体 Student 和课程实体 Class 的各个属性。

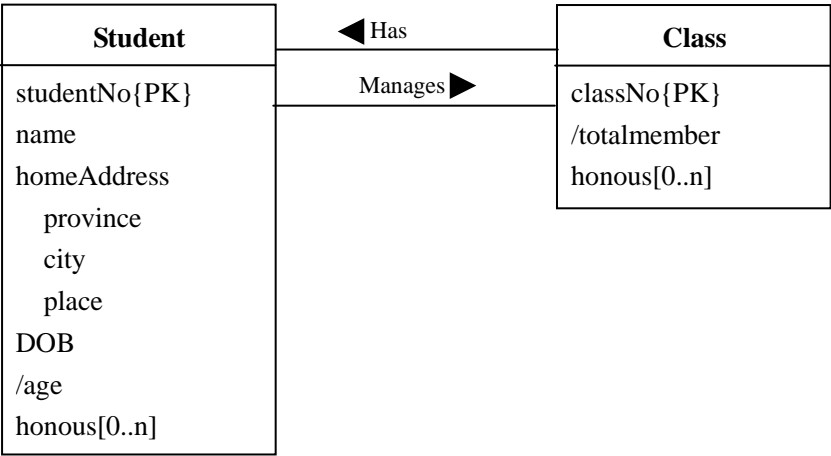


图 3.9 实体属性的表示

在图 3.9 中，Student 实体的主关键字为 studentNo，Class 实体的主关键字属性为 classNo，在图形中使用 PK 进行了标识。age 属性为 Student 实体的导出属性，totalMember 属性为 Class 实体的导出属性，使用 “/” 进行了标识。homeAddress 属性为 Student 实体的复合属性，通过缩进方式列出了该复合属性的每个组成部分。Student 实体与 Class 实体都具有一个多值属性 honous，它们可能取值为 0 个或者多个，通过[0..n]方式标识了其取值数目限制。

对于一些相对简单的数据库，有可能在 ER 模型图形化表示中列出每个实体的所有属性，然而对于复杂的数据库，如果在图形化表示中列出每个实体的所有属性，将使 ER 模型变得比较复杂和拥挤，这时可以在图形中只标识主要的属性，例如主关键字属性，或者不标识属性，而在设计文档中进行说明，从而使 ER 模型变得简单和清晰。

如果在数据库设计中使用了建模工具，则在 ER 模型设计中可以设计每个实体的所有属性，但在图形化显示中，可以根据需要设置显示或者不显示属性，或者显示哪些属性。

前面提到，联系也可能具有属性。在 ER 模型图形化表示中，对于联系的属性，采用与实体相同的表示方法，但为了区分带属性的联系与实体，一般用虚线将表示属性的矩形和联系关联起来，如图 3.10 所示。

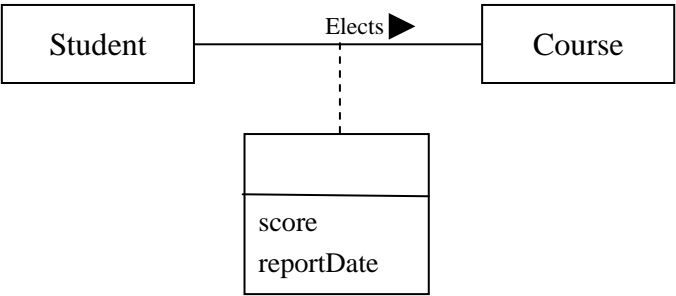


图 3.10 具有属性的联系的表示

在图 3.10 中，学生实体 Student 与课程实体 Course 之间具有选课联系 Elects，学生在选修某门课程之后最终将得到一个成绩，因此该联系将具有成绩属性 score 和得分日期属性 reportDate。

4) 强实体型与弱实体型

在前面的介绍中，实体都是独立存在的。然而，在 ER 建模过程中，根据需要也可能会出现这样一种实体，它依赖于其它实体而存在，称之为弱实体，也就是说，实体型可以分为强实体型和弱实体型两种，下面分别予以介绍。

强实体型指不依赖于其它的实体型而存在的实体型。我们前面在实体概念中介绍的实体型都指的是强实体型，例如，学生、课程与班级等实体型都属于强实体型。强实体型的一个显著特征是可以使用该实体型的主关键字唯一标识其中的每个实体，例如，学生的学号属性为主关键字，它能够唯一标识每个学生。

弱实体型指依赖于其它实体型而存在的实体型。考虑这样一个例子，假设学生在校期间可能获得各种奖励，也可能受到一些惩罚或者处分。在前面的例子中，曾经把奖惩记录作为一个多值属性设计，现在假设奖惩记录具有惩奖名称、惩奖级别与惩奖时间等属性，此时有必要将惩奖记录作为一个实体看待。不过该实体将是一个弱实体，因为它并不能够单独存在，而是依赖于学生实体的存在，即每个惩奖记录必须对应一个学生。图 3.11 显示了 Student 强实体与 Honous 弱实体之间的联系。弱实体型的一个特征是，仅使用该实体型本身的属性无法唯一标识该实体型中的每个实体。对于一个惩奖记录，它本身的三个属性惩奖名称、惩奖级别与惩奖时间都无法唯一标识每个实体，该实体只有与学生实体发生联系时才具有意义。弱实体有时也称为子实体、依赖实体或从属实体，强实体有时称为父实体、属主实体或支配实体。

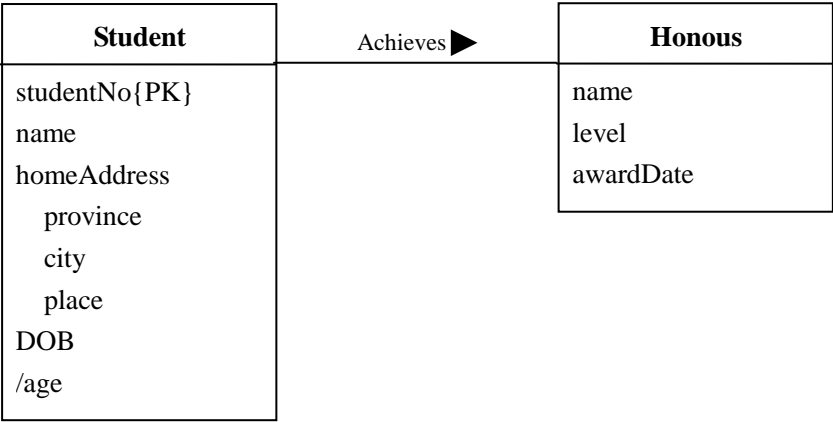


图 3.11 强实体与弱实体

在 ER 模型的建模过程中，弱实体并不是必要的，考虑上面的例子，如果设计人员把惩奖记录作为学生实体本身的一个属性，那么惩奖记录这个弱实体将不存在了，但是应该注意，惩奖记录作为学生实体的一个属性时，它是一个多值属性，同时也可能是复合属性。需要说明的是，把惩奖记录作为一个弱实体或者作为学生实体的一个多值属性，并不会影响逻辑数据库设计的结果，也就是说，对于两种不同 ER 模型情况，它们转化为逻辑模型之后是相同的。

5) 联系的约束

由于 ER 模型中使用实体型和实体型之间的联系型来描述用户需求，因此它只能描述实体型间联系的共性，而现实中实体个体之间的联系还具有个性，因此，仅依靠实体型间的联系仍然不足以准确

表达用户需求。例如，对于图 3.10 所示学生实体型与课程实体型之间的选课联系，仅表达了学生可以选修课程，却不能回答如下之类的问题：是否学生实体型中出现的每个学生可以选修多门课程？是否课程实体型中出现的每门课程可以被多名学生选修？也不能回答另外一类问题：是否学生实体型中出现的每个学生实体必须至少选修一门课程？是否课程实体型中出现的每门课程至少被一名学生选修？

其实，以上两类问题都是对实体型中个体参与联系的约束问题。在 ER 建模中，为了更准确表达用户需求，通常应对实体型间的联系型加以约束，这些约束用来反映实体型中实体个体在现实生活中参与联系的情况。例如，用约束来反映每个学生可以选修多门课程，每门课程也可以被多名学生选修等现实情况。

实体型间联系型的主要约束称为**多样性约束 (Multiplicity Constraints)**，它表示在一个特定的联系型中，一个参与联系的实体型中的某个个体，可能与另外一个参与联系的实体型中个体发生联系的数目或范围。由于数目或范围由最小数目和最大数目两个部分组成，因此，多样性约束实际上可以看作由两个独立部分组成的约束，即**基数约束 (Cardinality Ratio Constraints)**和**参与性约束 (Participation Constraints)**。

基数约束用来描述联系双方的实体型中，一方实体型中的实体可以与另一方实体型中某个实体发生联系的数目对比。基数约束有三种情况，分别为：一对一 (1:1)、一对多 (1:*) 和多对多 (*:*)。

例如，对于学生实体 Student 与班级实体 Class 之间的 Manages 联系，该联系应该是一个一对一的联系，它表示班长与班级之间是一一对应的关系，即一个班长只能管理一个班级，一个班级也只能有一个班长；对于班级实体 Class 与学生实体 Student 之间的 Has 联系，该联系应该是一个一对多的联系，它表示一个班级可以拥有多名学生；对于学生实体 Student 与课程实体 Course 之间的选课联系，该联系应该是一个多对多的联系，它表示一名学生可以选修多门课程，一门课程也可以被多名学生选修。

参与性约束用来描述联系双方的实体型中，每一方实体型中的实体参与联系的最小数目，即是否每一方实体型中的所有实体都参与了联系。参与性约束有两种情况，分别为：**强制参与 (Mandatory Participation)**和**可选参与 (Optional Participation)**。

强制参与约束表示参与联系的一方实体型中的所有实体都参与了联系，而可选参与则表示参与联系的一方实体型中的部分实体参与了联系，即允许部分实体不参与联系。

例如，对于班级实体 Class 与学生实体 Student 之间的 Has 联系，在该联系中，如果学生实体中的任一学生都必须属于某一个班级，也就是说学生实体中的每个学生个体都必须参与该联系，那么学生实体在该联系中是强制参与的。同样，在该联系中，如果规定每个班级至少拥有一名学生，那么班级实体在该联系中也是强制参与的。

联系的多样性约束是由用户根据现实情况的需求确定的，并不是数据库设计人员主观规定的。例如，对于上述 Has 联系，若用户规定每个班级至少拥有一名学生，因此，班级实体在该联系中也是强制参与的；如果用户根据其应用环境允许某些班级可以暂时没有学生，那么班级实体在该联系中将变

为可选参与的。

在 ER 模型的图形化表示中,参与联系双方实体的多样性约束标识于联系双方实体附近,如图 3.12 所示。

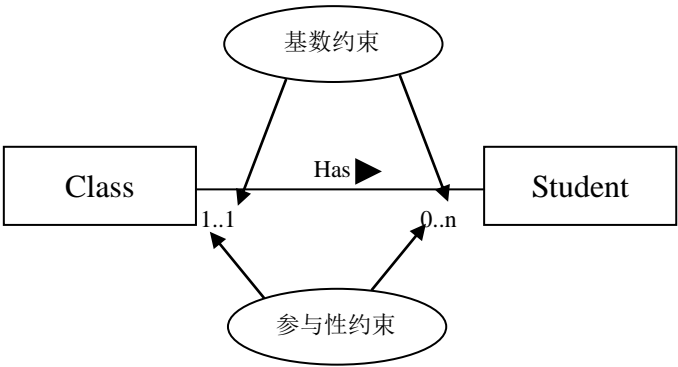


图 3.12 联系约束的图形化表示

在图 3.12 中,班级实体 Class 与学生实体 Student 之间具有联系 Has,对于基数约束,Has 联系是一个一对多的联系,其中实体 Class 是一方,实体 Student 是多方;对于参与性约束,Class 实体在该联系中是可选参与的,即允许某些班级暂时没有学生,Student 实体在该联系中是强制参与的,即任一学生必须属于某个班级。

在介绍了联系的多样性约束之后,我们再来考虑一下前面提到的强实体与弱实体之间联系约束情况。同样考虑学生强实体与惩罚记录弱实体之间的联系,在现实情况中,对于学生实体,并不一定每个学生都有惩罚记录,一个学生也可能会有多个惩罚记录。那么对于惩罚记录实体呢?应该说,每个惩罚记录都应该对应一个学生,而且只能对应一个学生。也就是说,学生实体与惩罚记录实体之间的联系是一个一对多的联系,并且学生实体在联系中是可选参与,而惩罚记录实体在联系中是强制参与的。

由以上分析可知,一般情况下,对于强实体与弱实体之间的联系,强实体在联系中是一方,弱实体是多方;强实体在联系中是可选参与的,而弱实体是强制参与的。

对于高于二元联系的多元联系,它们的多样性也同样具有多样性约束,只不过要复杂一些。多元联系的多样性约束表示在一个 n 元联系中,当其它 (n-1) 个参与实体型的值确定之后,它可能参与联系的数目或范围。由于三元和三元以上联系在数据库建模过程中较少使用,在此不再详细论述,仅通过图 3.13 给出一个三元联系例子,其中标识出联系各方实体型的多样性约束,表达了用户潜在的需求,读者可以自行分析。

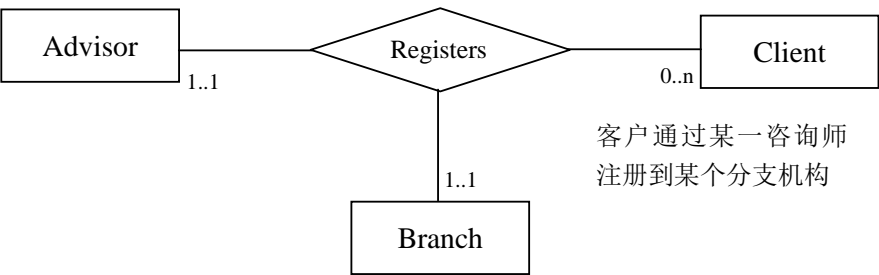


图 3.13 多元联系的约束表示

3. 2. 2 ER 模型建模过程中经常遇到的问题

ER 模型是通过实体以及实体间的联系来描述现实世界，而现实世界中实体间的联系有时是非常复杂的，并且在不同环境和场所或者领域，联系的表现形式也不一样。另外，一个实体也可能与多个实体发生联系，这些联系有些是直接联系，有些是间接联系（可以通过与另外一个实体的联系得到的联系），这种情况下，就需要特别考虑。

针对以上联系发生的情况，在 ER 模型的建模过程中，如果设计人员对用户需求理解不够充分，或者遗漏了实际中某些情况的发生，将会导致一些错误的建模结果，而这些结果看起来似乎正确。在这里主要讨论两类常见的问题：扇形陷阱（fan trap）和深坑陷阱（chasm trap），并讨论如何在 ER 模型中发现并解决这些问题。扇形陷阱和深坑陷阱都属于连接陷阱（connection trap），它们都是由于建模过程中对实体间连接次序错误或者连接遗漏导致的。

1) 扇形陷阱

扇形陷阱是指在 ER 模型中，某些实体之间的通路（pathway）是不明确的情况。为了说明这一问题，考虑图 3.14 所示的模型，该模型用来反映这样一个事实，一个学院 College 有多个学科方向或系 Department，学院可以聘有多名教师 Teacher。



图 3.14 扇形陷阱的例子

该模型看起来似乎没有问题，但是当我们想知道某位教师到底属于哪个学科方向或系时，问题就产生了，因为根据该模型中，很可能无法确定某些教师属于哪个学科方向或系。为了具体展现问题的存在，图 3.15 根据该模型给出了一些实体个体以及它们之间的联系情况。在该图中，当我们试图回答“T101 教师属于哪个系”这个问题时，便无法得到确切答案，也就是说教师与学科方向或系之间的通路是不明确的，产生了扇形陷阱问题。

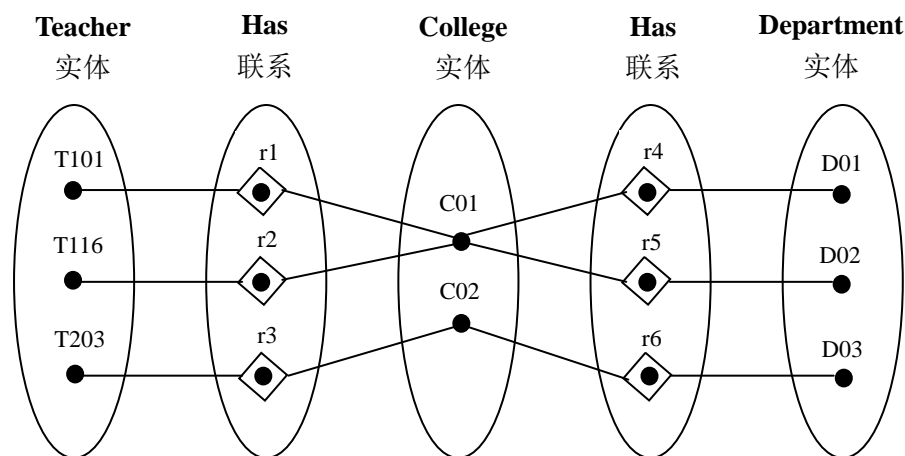


图 3.15 扇形陷阱的实体个体之间联系

对于 ER 模型中存在的扇形陷阱问题应该如何解决呢？仔细分析图 3.14 的 ER 模型，不难发现，该模型中实体之间的联系次序是不合适的，它与现实情况并不吻合。在现实中，教师应该是与学科方向发生直接联系，学科方向和学院之间也是直接联系，教师与学院之间的联系应该属于间接联系，它可以通过教师与学科方向的联系达到与学院联系的目的。也就是说该模型中实体间的联系次序是不合适的。为了解决这一问题，只要把模型中的学院与学科方向的位置互换，从而改变模型中实体之间联系次序，重新构建模型即可，如图 3.16 所示。根据该模型，图 3.15 所表示的部分个体之间的联系将如图 3.17 所示，该图清楚地标识了哪些教师属于哪个学科方向。

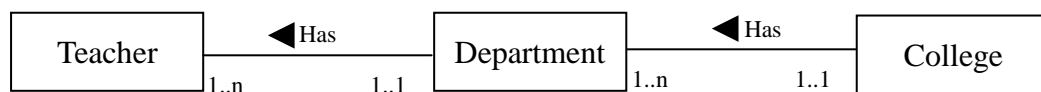


图 3.16 扇形陷阱的解决方法

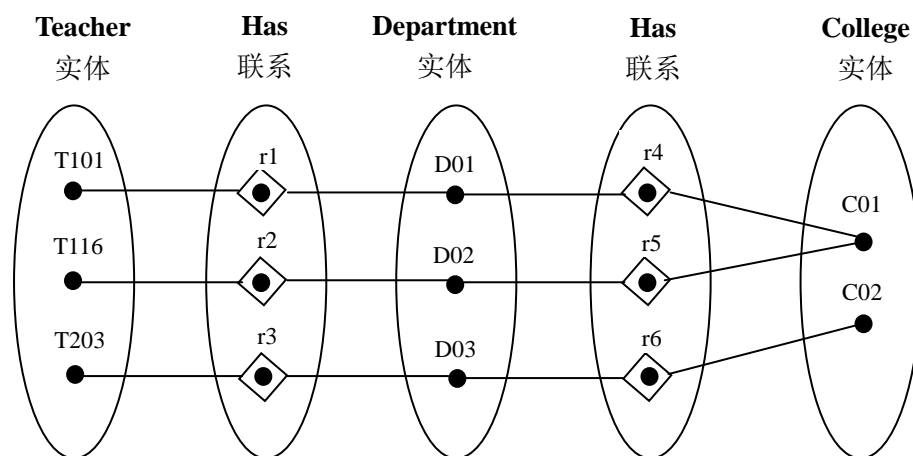


图 3.17 扇形陷阱解决后实体个体之间联系

2) 深坑陷阱

深坑陷阱是指在 ER 模型中，某些实体型之间存在联系，而这些实体型中的部分实体却不存在相应的通路。为了说明这一问题，继续考虑在扇形陷阱讨论中，经过我们改造之后的图 3.16 所示的模

型，该模型能够正确反映这样一个事实，即一个学院有多个学科方向或系，系聘有多名教师，该模型消除了原来模型中存在的扇形陷阱问题。图 3.16 所示的模型还反映了这样一个事实，即教师必须并且只能属于某一个系，现在如果用户需求有细微变化，允许某些教师暂时不属于任何系，即该模型变为图 3.18 所示。图 3.18 与图 3.16 的唯一差别就是教师在与系的联系中，它的参与性约束发生了变化，图 3.16 反映的事实是教师必须并且只能属于某一个系，而图 3.18 反映的情况是允许部分教师暂时不属于任何系。



图 3.18 深坑陷阱的例子

该模型反映了用户需求的变化，似乎没有问题，但是如果仔细考虑，就会发现，用户的这一需求变化，影响的并不仅仅是教师实体在联系中的参与性问题。在图 3.16 的模型中，教师实体型中的每个个体都参与了教师实体型与学科方向之间的联系，因此可以确定每个教师所在的学科方向，而学科方向又必须属于学院，因此所有教师都属于学院。现在由于需求变化，有部分教师可以不参与教师实体型与学科方向实体型间这一联系，那么问题是没有参与这一联系的那部分教师是否仍属于学院呢？如果回答是肯定的，那么在该模型就存在问题了。图 3.19 根据该模型给出了一些实体个体以及它们之间的联系情况。在该图中，当我们试图回答“T006 教师属于哪个学院”这个问题时，便无法得到确切答案。也就是说根据该模型，教师与学院应该存在某种联系，但部分教师却不能够通过学科方向与学院联系起来，这就是深坑陷阱问题。

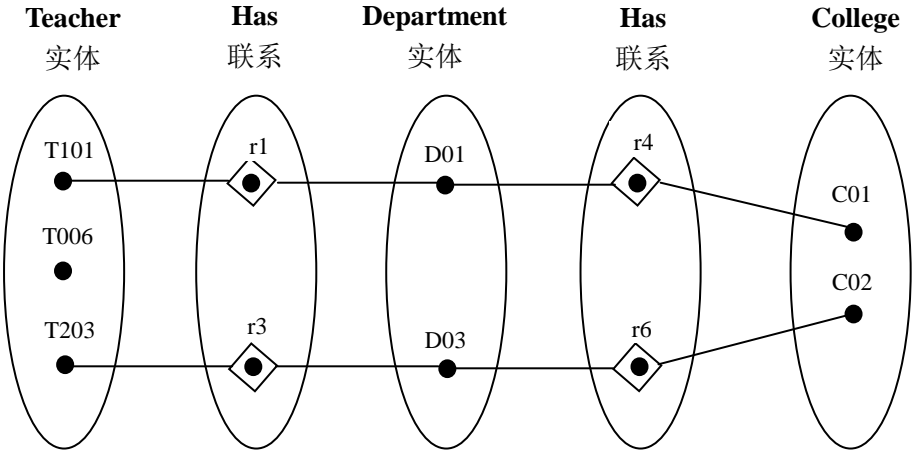


图 3.19 深坑陷阱实体个体之间的联系

深坑陷阱问题导致了部分教师无法通过学科方向与学院联系起来，他们与学院之间的联系被遗漏了，因此解决深坑陷阱问题的方法就是重新标识被遗漏的联系，如图 3.20 所示。在图 3.20 中，通过增加教师与学院间的一个联系，从而使得所有教师都能间接或直接与学院发生联系，解决了深坑陷阱的问题，图 3.19 所示的部分实体个体之间的联系如图 3.21 所示，图 3.21 中消除了图 3.19 中无法确定 T006 教师属于哪个学院的问题。

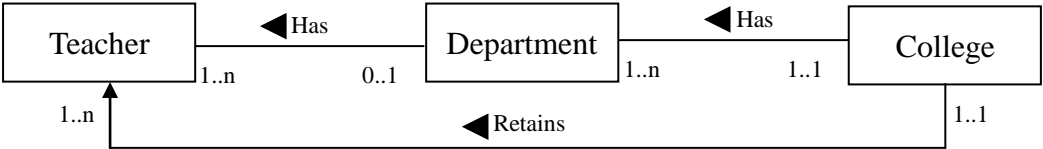


图 3.20 深坑陷阱的解决

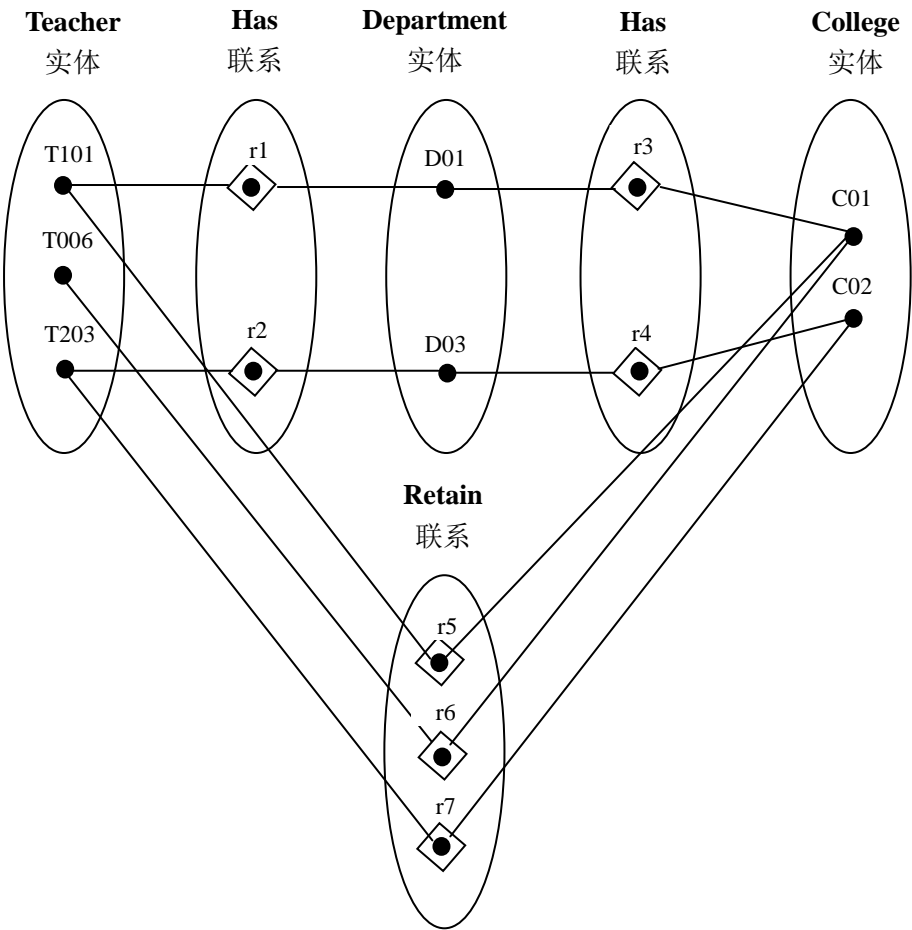


图 3.21 深坑陷阱解决之后实体个体之间的联系

以上通过具体案例分析了扇形陷阱和深坑陷阱产生的原因，并给出了具体的解决方法。通过以上分析，我们可以对扇形陷阱和深坑陷阱做如下一般性的总结：

- (1) 在一个 ER 模型中如果存在一个实体与其它实体之间存在两个或更多的一对多联系，很可能存在扇形陷阱问题，扇形陷阱问题可以通过改变实体间的联系次序，从而重新构建模型解决；
- (2) 在一个 ER 模型中如果在实体联系的通路存在一个或者多个参与性约束最小值为零的情况，很可能存在深坑陷阱问题，深坑陷阱问题可以通过重新添加被遗漏的联系解决。

3.2.3 概念数据库设计过程

通过上一小节内容的介绍，我们知道 ER 模型是当前最为常用的概念数据库设计技术，由于 ER 模型具有非技术性和无二义性的特点，使得它能够成为最终用户和数据库设计人员之间的交流桥梁，因

而也得到了普遍的应用。然而 ER 模型仅仅是一项技术,或者方法,但是对于解决实际工程问题,仅仅具有技术或者方法是不够的,还需要过程的支持,甚至还需要工具的支持,这些内容都属于方法学的范畴。

方法学又称方法论,在哲学意义上,它是一门学问采用的方法、规则与公理,或者定义为一门学问采用的一种特定的做法或一套做法。对于软件来讲,软件方法学(SoftWare Methodology)是以方法为研究对象的软件学科,通常把在软件生命周期全过程中使用的一整套技术的集合称为软件方法学。因此,软件方法学涉及了指导软件设计的原理和原则,以及基于这些原理、原则的方法、技术和过程,其关注的中心问题是如何设计正确地和高效率地设计软件。

在普遍的工业行业中,特别是在加工生产中,方法学一直占有重要地位。在机械加工行业中,零件的加工有严格的工序和过程要求,操作必须按照预先制定工序和过程要求,才能高效生产出合格产品。在食品加工行业中也有严格个工序和过程要求,这样,生产出来的食品才营养丰富并可口。这些都是方法学在行业中的应用体现。如果没有完善的方法学,在机械加工中,即使采用相同的材料,每次加工的零件质量可能都不同;在食品加工中,即使采用相同的原料,每次生产的食品口味也会大不一样。更重要的是,方法学不但保证产品加工的质量,而且它能够提高产品生产的效率,因为方法学不仅仅是技术,它是是在长期生产过程中总结出来的、经过实践检验的、并且不断完善和发展的体系。

综上所述,方法学涵盖了三个方面的内容,即方法、过程和工具,在工业生产和工程中占有重要地位。数据库设计作为一项工程,方法学在其中同样起着重要作用,因此,学习数据库设计不仅仅是学习数据库设计技术,技术仅仅是方法学中的一个方面,还要学习数据库设计过程以及支持工具。

针对数据库设计来讲,方法就是完成数据库设计各项任务的技术,是回答“如何做”的问题,上一小节中介绍的 ER 模型就是一种概念数据库设计技术;过程是为了获得高质量的软件所需要完成的一系列任务的框架,它规定了完成各项任务的工作步骤,本小节接下来将详细介绍概念数据库设计过程;工具是为方法的运用提供自动的或半自动的软件支撑环境,本章将在后面进行介绍。

概念数据库设计是在不考虑任何物理因素的情况下,构建企业信息模型的过程,上一小节已经介绍了在该阶段构建模型的技术—ER 模型技术,对于一个简单的系统,可能不需要特定的步骤就能够很快得到 ER 模型,然而对于一个真正的企业信息系统,一般是比较复杂,在需求收集和分析完成之后,如果没有一个很好的步骤构建 ER 模型,往往会陷入泥潭,或者得不到满足要求的 ER 模型,或者效率低下,遗漏错误频繁发生。为了高效地完成概念数据库设计,从而得到满足要求的 ER 模型,本小节给出工程中构建 ER 模型一般采用的步骤和过程。

本章在第一小节曾经提到,一个系统中可能存在多个用户视图,在需求分析中通常可以采用三种方法处理,它们分别是视图集中、视图集成或者两种方法的混合。如果采用视图集成的方法,针对需求分析得到的多个用户的局部视图,数据库设计应该针对每个局部视图进行,最终,针对这些局部视图的设计结果将合并为全局数据库模型,合并过程一般在局部视图的逻辑数据库设计完成之后进行。下面给出的概念数据库设计过程和步骤适合于针对每个局部视图的概念数据库设计。

1) 标识实体型

本步骤的主要目的是根据用户需求说明书,标识其中主要实体型。标识实体型一般有两种常用的

方法。

一种方法是审查用户需求说明书，标出其中所有名词和名词短语（比如，学生、课程、教师、学号、学生姓名、课程编号、课程类型等），同时也要注意一些概念上的对象（比如，成绩等）。在这些名词和名词短语中，要区分表示对象的名词和仅仅描述对象性质的名词。例如，学号和学生姓名仅仅是描述学生具有的某些性质的名词，因此它们应该为学生实体的属性，而不应该作为单独实体。

标识实体的另一种方法是根据用户需求和数据库应用环境，查找哪些客观存在的对象。例如，教师是一个实体，该实体在教学管理系统中是客观存在的，即使我们不知道它们的名称、职务等。某些情况下，系统中客观存在的对象在用户需求说明书中可能没有明显体现，这时，应该在用户的帮助下完成这个过程。

用户需求说明书中实体的表达方式可能各不相同，有时会给确定实体带来一定困难。一种情况是，用户在需求说明书中经常会用具体的某个实体来说明需求，例如，直接提到某个学生的名字而不是广义上的学生，提到某个具体角色院长等，这时要注意从需求中抽象出合适的实体。另一种情况是，用户在需求中经常会使用同义词来表示同一个实体，例如，在学生视图中的教师与教师视图中的职员都指教师这一实体，这时需要根据上下文确定需求中出现的同义词是否为同一实体，并通过用户确认。

在标识出实体型后，应为其指定有意义并且明了的名称，并遵照相关标准在文档中记录这些实体型及其描述。如果某个实体可能有多个名字，应将它们定义为同义词或别名，并在文档中记录下来。

2) 标识联系型

本步骤的目的是标识实体型之间存在的重要联系。在上一步骤标识实体过程中，我们曾经提到其中的一种方法是查找用户需求说明书中出现的名词和名词短语，对于标识联系，用户需求说明书同样具有帮助，通常，联系在用户需求说明书中表现为动词或动词词组。例如：学生选修课程，班级拥有学生，班长管理班级等，其中选修、拥有和管理都是联系。在确定联系之后，特别应该注意联系约束的标识，因为用户需求说明书中对联系约束的描述可能并不明显，有时可能还需要设计人员根据用户系统使用要求先行标识并向用户确认。比如对于上面提到的班级拥有学生联系，用户需求中可能没有明确描述任一班级是否必须至少拥有一名学生，或者也没有明确描述任一学生是否必须属于一个班级，而这些问题都涉及到班级拥有学生这个联系的约束问题，设计人员必须根据用户使用要求进行设计并向用户确认。

大多数情况下，联系是二元的，但是在标识联系的过程中，应该注意单个实体的递归联系以及多个实体间多元联系的出现。例如，某些课程可能具有一门或多门先修课程要求，该先修联系便是课程实体与自身之间的一个递归联系。

在标识联系过程中，设计者要仔细分析，尽可能标识出用户需求说明书中显式和隐式联系，原则上应该检查每一对实体，找出它们之间存在的联系，对于一个具有上百个实体的大型系统来说，这将是一项复杂的工作，系统分析者和数据库设计者需要具有充分的耐心和仔细。如果本步骤遗漏了某些联系，这些遗漏的联系在后面验证局部概念模型（第7步）时应该很容易发现。

在标识出联系型后，同样应为其指定有意义并且明了的名称，并遵照相关标准在文档中记录这些联系型及其描述，尤其要清楚描述联系的多样性约束。

在标识出实体和联系之后，使用可视化的方式描述系统能够使得设计结果更容易被理解，因此，在该步骤可以采用前面介绍的 ER 模型的图形化表示方式进行描述。

该步骤完成之后，应该对得到的初步 ER 模型进行必要检查，审查模型中的实体以及它们之间的联系是否满足用户需求说明书的要求，特别对于没有参与任何联系的实体，两个实体间存在多个联系，递归联系以及多元联系等情况，需要仔细检查和确认。

3) 标识每个实体型和联系型的属性

标识属性的方法类似于标识实体，通过查找用户需求说明书中的名词或者名词短语，如果这些名词或者名词短语描述的是实体或者联系的一种性质或特征时，即可作为属性标识。用户需求说明书中可能仅仅描述实体的某些主要属性，对于实体的确切属性，有时需要设计人员与用户沟通和交流进行确定。

在标识属性过程中，应该注意属性的类别和特性，比如前面介绍的简单属性或者复合属性，单值属性或者多值属性以及导出属性，它们将决定在逻辑数据库设计中的处理方式。

标识了属性之后，为它们指定有意义并且明了的名称，并遵照相关标准在文档中记录这些属性及其描述，这些记录应该包含的信息包括：属性名称、属性描述、属性的数据类型和长度、属性是否为复合属性（如果为复合属性，应给出组成它的简单属性）、属性是否为多值属性（如果为多值属性，应给出可能取值范围），属性是否为导出属性（如果为导出属性，应描述该属性的计算方法），属性是否具有默认值（如果有应指定默认值）。

属性标识之后，应确保每个属性都与实体或者联系相关联。

4) 确定属性域

本步骤的目的是为 ER 模型中的所有属性确定域。域是值的集合，一个或者多个属性可以从中取值。例如，对于学号属性，可以定义一个域为 8 位字符长度的字符串，字符串前面两位为字母，然后便可以指定学号属性的数据类型和长度为该域。

属性域确定之后，也应遵照相关标准通过文档中记录这些属性域及其描述，并对前面属性文档中的属性数据类型和长度信息进行修订，使用属性域替代。

5) 确定候选关键字以及主关键字

该步骤主要考虑为实体确定候选关键字，并选择其中一个作为主关键字（参见 2.1.4 节）。在从候选关键字中选择主关键字时，实际并不是随意选取，一般遵循一些基本的原则，例如：

- 选取用户经常使用的候选关键字；
- 选取属性组合最少的候选关键字；
- 对于字符属性，选取具有最少字符数属性的候选关键字；
- 对于数字属性，选取数值范围较小的候选关键字；

在确定关键字的过程中，应该结合具体应用环境和应用范围进行选取和确定，另外，如果 ER 模型中存在弱实体，可能暂时无法确定这些弱实体的主关键字，在逻辑设计过程中，通过把弱实体所依赖的主实体的主关键字属性放入弱实体之后才能确定。

确定实体的候选关键字和主关键字属性之后，也应在相应文档中进行记录，一般可以在前面的实

体文档中进行记录。

6) 检查模型中的冗余

该步骤通过检查前面得到的局部概念数据模型，以确定其中是否有冗余，有则删除它们。该步骤检查主要有两个方面：一方面是检查冗余的实体；另一方面是检查冗余的联系。

在检查模型中可能存在的冗余实体时，主要应该注意哪些存在同义词的实体，检查这些实体是否在模型中以不同的角色出现多次，如果有则应消除。另外，需要重点关注存在一对一联系的双方实体，检查它们是否应为同一实体的不同展现，若一对一联系双方实体的确是同一实体，应予以合并。

在检查模型中可能存在的冗余联系时，如果某个联系表示的信息能通过其它联系获取，则该联系是冗余的，冗余的联系是不必要的，应该予以消除，否则这些冗余可能会给后续设计带来问题，或者导致最终数据库的冗余。需要注意的是，两个实体间存在多个路径并不意味着存在冗余的联系，因为多个路径完全可能表示两个实体间的不同关联。为了说明联系冗余的问题，考虑图 3.22 所示的情形。

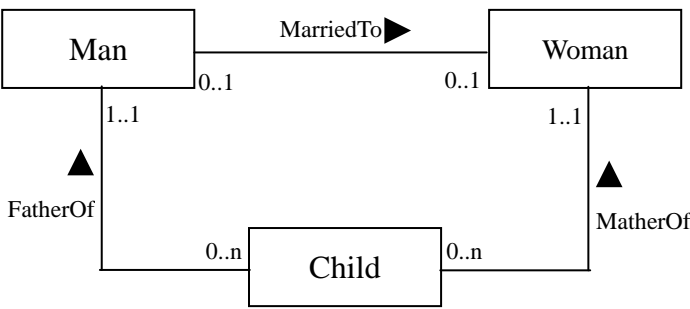


图 3.22 冗余联系的例子

在图 3.22 所示的模型中，很明显，在 Man 和 Child 之间存在两个路径：一个通过直接联系 FatherOf，另一个通过间接联系 MarriedTo 和 MatherOf。然而这并不能认为联系 FatherOf 是多余的，原因是，模型中 Man 和 Women 是一一对一的联系，也就是父亲当前只能有一个妻子，然而父亲可能会由于先前的婚姻而拥有孩子，而这个孩子是无法通过两个间接联系 MarriedTo 和 MatherOf 与父亲发生联系的，因此 FatherOf 并不能取消。但是，如果应用系统中规定一旦一个男性与一个女性通过婚姻组成家庭之后，他们婚姻前的孩子将一起归入新的家庭，通过孩子可以唯一确定当前父亲和母亲，那么 FatherOf 联系可能是不必要的。因此，在确定模型中联系冗余时，需要考虑现实情况，也要考虑系统将来使用中的要求。

7) 验证局部概念模型

本步骤的目的时检查模型以保证模型能够支持用户视图所需要的事务，一般采用事务路径的方法。事务路径的方法从两个方面检查模型，一方面是描述用户视图中要求的每个事务的数据需求，以检查模型是否提供了所有事务的数据需求；另一方面是在 ER 模型中直接表示每个事务的数据需求路径，以确认用户视图中要求的每个事务都能从模型中顺利得到所需数据。

采用事务路径的方法，设计人员能够形象地表示出模型中哪些数据与哪些事务相关，如果模型中存在不被任何事务使用的部分，那么需要质疑该部分在模型中存在的目的。另外，如果模型包含了事务需要的数据，但是却没有获取事务所需数据的正确路径，那么需要质疑模型中是否遗漏了某些联系

或者实体。

对于大型系统，验证局部概念模型是非常复杂的，但是必须认真进行，因为如果省略掉这个步骤或者延至以后进行，发现模型存在的错误将更困难，解决问题花费的代价也将更大。

8) 与用户共同审查局部概念模型

概念数据模型的结果包括 ER 图和描述数据模型的支持文档。本步骤的目的是与用户一起复查通过以上步骤得到的局部概念数据模型以保证该模型准确描述了用户视图。在复查过程中如果发现任何异常问题，设计人员必须对模型进行修正，这可能需要重复前面的步骤，直到模型得到用户的认可。

在本书采用的 Teaching 数据库系统案例中，有两个用户视图，它们分别为学生视图和教师视图，图 3.23 (a) 和 (b) 分别表示了针对这两个用户视图进行概念数据库设计得到的 ER 模型图。

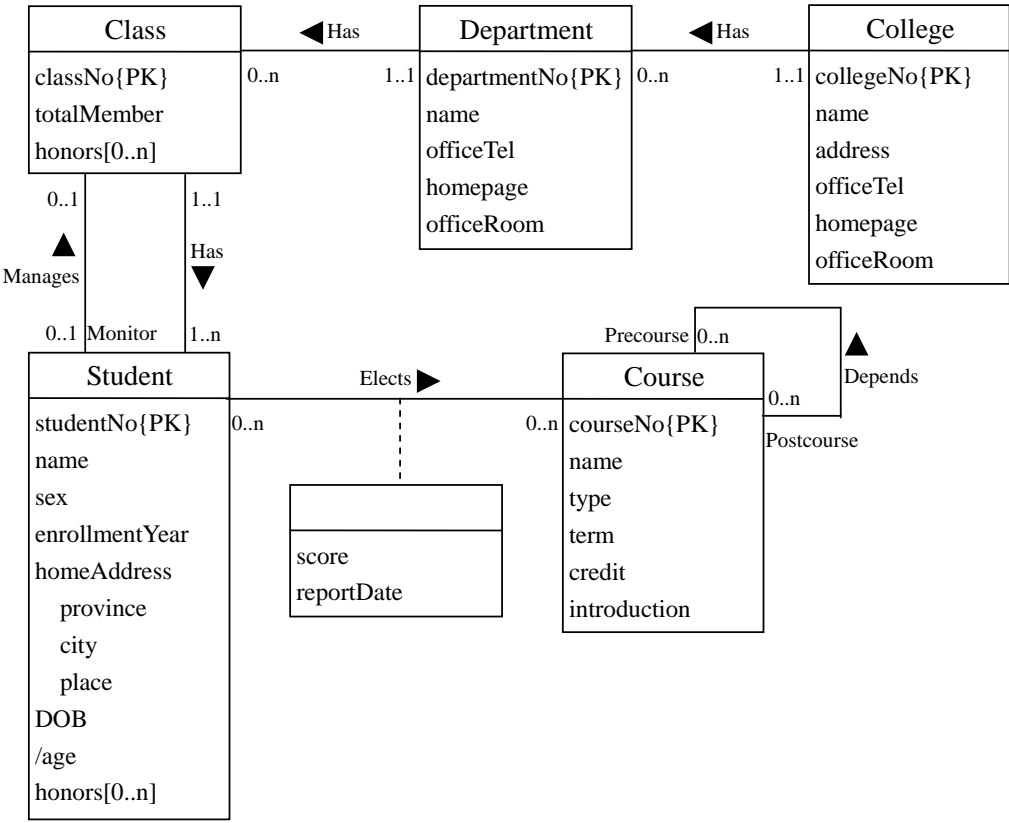


图 3.23 (a) Teaching 案例 Student 视图的 ER 模型图

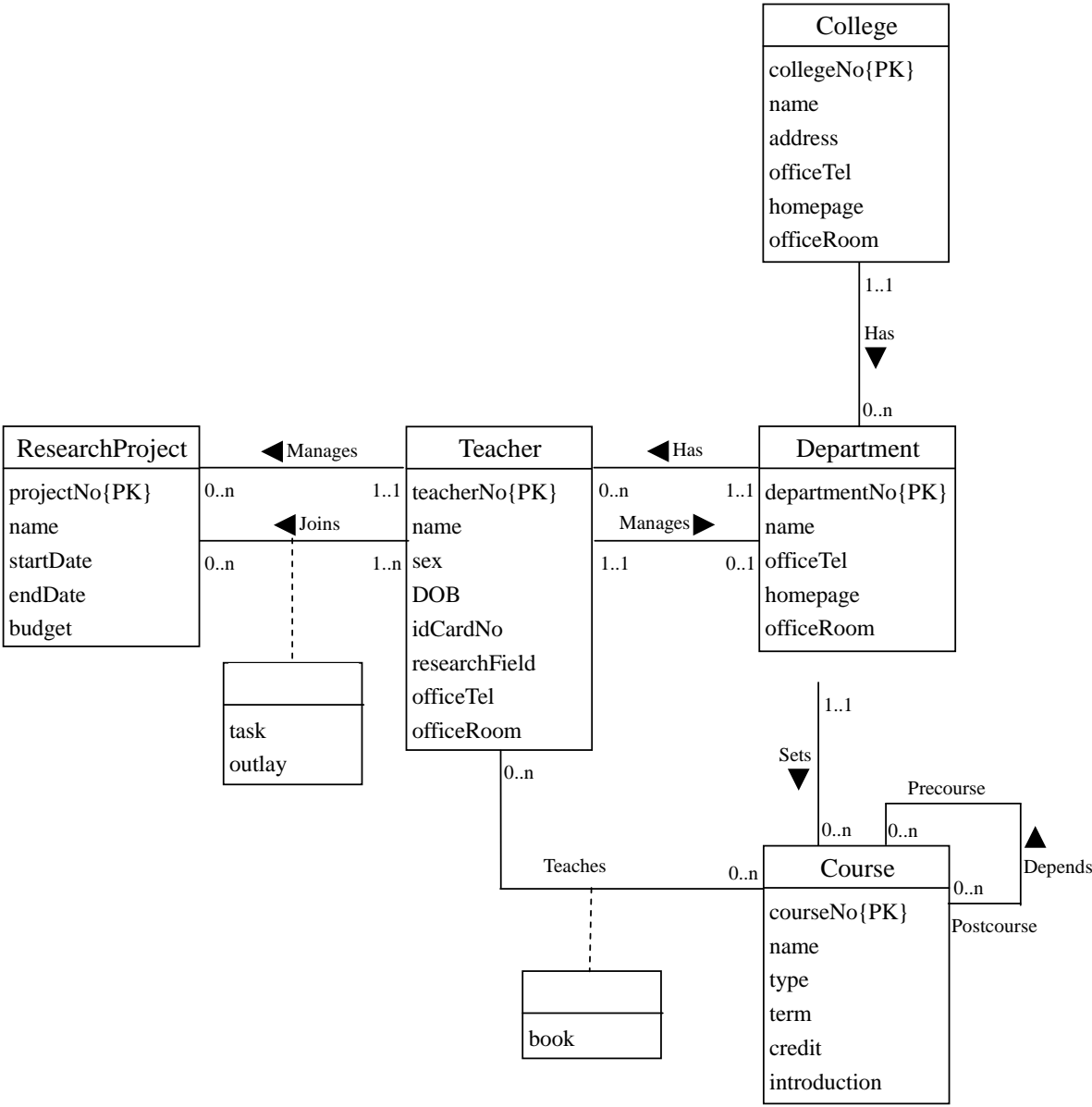


图 3.23 (b) Teaching 案例 Teacher 视图的 ER 模型图

3.3 逻辑数据库设计



内容提要

在 3.1 节中，我们把数据库设计分为三个主要阶段进行，它们分别是：概念设计、逻辑设计和物理设计。这三个设计阶段中的每个阶段将产生不同的结果模型，具有不同的用途和目的。在 3.2 节，我们从技术、过程两个方面详细论述了数据库设计的第一个阶段—概念数据库设计，本节将继续论述数据库设计的第二个阶段—逻辑数据库设计。本节主要包含如下内容：

- ER 模型向关系模型的映射；
- 逻辑数据库设计过程。

上一节论述的概念数据库设计是在不考虑任何物理因素的情况下，构建企业信息模型的过程，该阶段的主要目的是充分挖掘系统对数据的需求，因而该阶段建立的模型应该容易为设计人员和用户所理解，能够成为设计人员和用户之间交流的桥梁。然而，数据库设计的最终目标是要在数据库管理系统中实现模型，而概念数据库设计的目的并不是针对具体数据库管理系统进行的，因此该阶段的设计结果并不能够为数据库管理系统所支持。为了让概念数据库设计结果能够为数据库管理系统所支持，必须进行逻辑数据库设计。

逻辑设计是将概念设计阶段得到的结果转化为目标 DBMS 所支持的数据模型的过程。在概念数据库设计完成后，我们得到的结果是概念数据模型（一般为 ER 模型）和支持文档。而目前广泛应用的 DBMS 大部分为关系 DBMS，支持关系模型，因此逻辑设计的结果应该为关系数据模型。所以，逻辑数据库设计的主要工作就是如何将 ER 模型转化为关系模型。在 ER 模型中，我们通过实体以及实体之间的联系表达了用户的需求，而在关系模型中，数据结构非常单一，只有关系，因此，ER 模型中的实体以及实体之间的联系最终都将映射为关系模型中的关系以及关系之上的约束。

上一节论述的概念数据库设计过程是针对每个用户的局部视图进行的，如果系统中存在多个用户视图，那么需要针对每个用户视图分别创建局部概念数据模型，因此，概念数据库设计的结果将是多个局部 ER 模型以及支持文档，在逻辑数据库设计阶段，同样要针对每个局部 ER 模型进行，它们的过程是相同的，只不过如果系统存在多个用户视图，概念数据库设计结果将得到多个 ER 模型，它们分别转化为逻辑模型之后，也将得到多个局部逻辑数据模型，最后需要对这多个局部逻辑数据模型进行合并，从而得到全局逻辑数据模型。下面给出工程中将概念设计阶段得到的 ER 模型映射为逻辑模型通常的方法和步骤，在映射过程中仍采用 Teaching 案例进行说明。

1) 对于 ER 模型中的多对多二元联系

如果在 ER 模型中存在一个多对多的二元联系，可以通过设置一个中间实体，从而将该多对多的二元联系分解为两个一对多的二元联系。新增加的中间实体一般为弱实体。考虑图 3.24(a) 显示的案例，它表示了学生实体 Student 与课程实体 Course 之间的多对多二元联系选课 Elects，该联系表示一名学生可以选修多门课程，一门课程也可以被多名学生选修，同时该联系具有成绩 score 和 reportDate 两个属性，表示学生选修课程之后，将在某个时间得到一个成绩。对于该多对多二元联系，可以通过设置一个中间实体 CourseReport，从而将 Elects 联系分解为新设置实体与原来两个实体间的两个一对多二元联系，如图 3.24(b) 所示。其中，新增加实体 CourseReport 为一个弱实体，因为它的存在依赖于 Student 实体和 Course 实体。

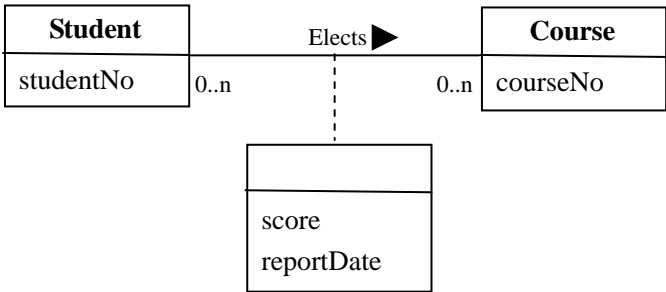


图 3.24（a）Student 实体与 Course 实体间的多对多联系

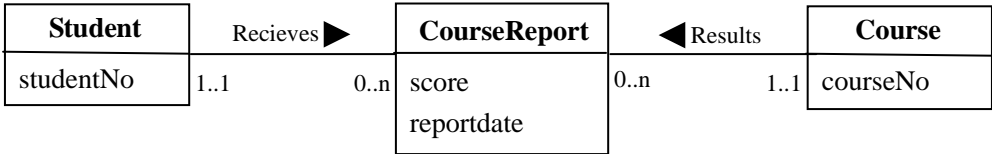


图 3.24（b）多对多联系的转化

2) 对于 ER 模型中的多对多递归联系

递归联系是一种特殊的联系类型，它是一个实体类型和它本身之间发生的联系，递归联系有三种类型，它们分别为：一对一递归，一对多递归和多对多递归。对于一对一递归和一对多递归两种联系处理相对简单，后面讨论，在本步骤首先处理多对多递归联系。

多对多递归联系也是一种特殊的多对多联系，只不过联系双方是同一个实体，对于它可以采用和一般多对多联系同样的方式处理，只是处理前需要稍作变化。例如，考虑图 3.25(a)显示的课程实体 Course 之上的多对多递归联系 Depends，它表示了一门课程可能具有多门先修课程要求。首先将该多对多递归联系转化为两个实体间的一般的多对多递归联系，如图 3.25(b)所示，然后对于该一般的多对多联系，其处理方式同第一步中的处理方式，通过引入中间（弱）实体 CourseDependence，把多对多联系分解为两个一对多联系，如图 3.25(c)所示。最后，由于分开的两个 Course 实体为同一个实体，需要重新合并这两个实体，如图 3.25(d)所示。

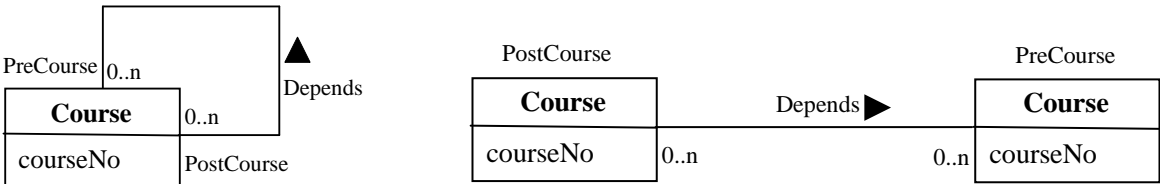


图 3.25（a）Course 实体之上的递归联系

图 3.25（b）递归联系的分解

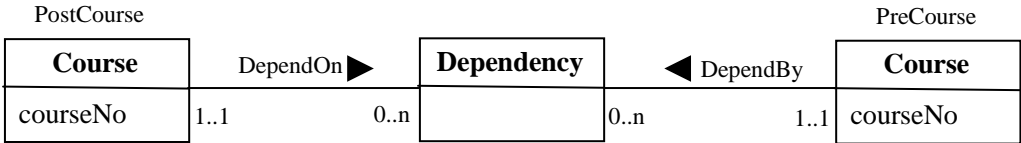


图 3.25（c）对对多联系的转化

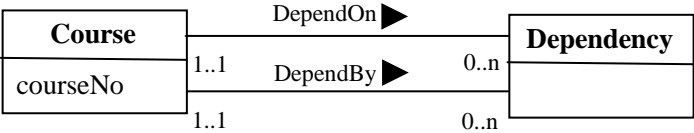


图 3.25（d）多对多递归联系合并

3) 对于 ER 模型中的复杂联系

复杂联系是三个或更多实体型间的联系，如果在 ER 模型中存在这样的复杂联系，可以通过设置一个中间实体，该新设置实体一般为弱实体，从而把该复杂联系分解为原有实体与新设置实体间的多个一对多二元联系。例如，考虑图 3.26(a)所示的三元联系 Registers，在前面已经讨论过，可以通过设置(弱)实体 Registration，从而将三元联系 Registers 分解为三个一对多二元联系，如图 3.26(b)所示。

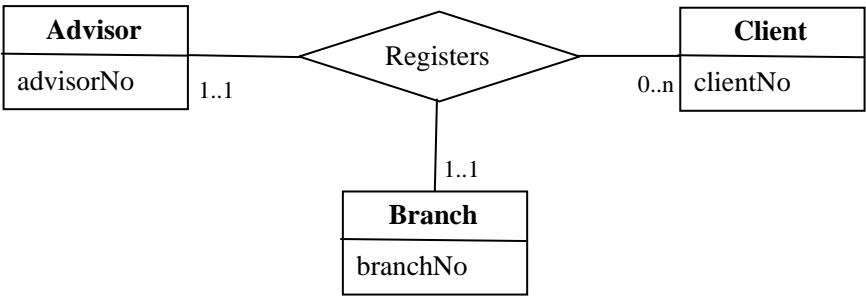


图 3.26 (a) 三元联系

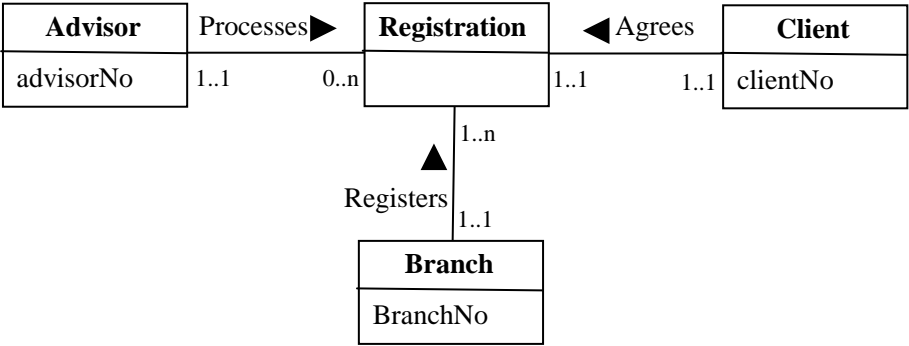


图 3.26 (b) 三元联系的转化

对于三元以上的复杂联系类型，可以采用同样的方式处理。

4) 对于 ER 模型中的多值属性

多值属性可以为一个实体的某个属性保存多个值，如果 ER 模型中存在这样的多值属性，需要将这样的多值属性分离为一个实体。例如，对于学生实体 Student 的奖惩记录属性 honors，由于学生在校期间可能获得多个奖惩，因此该属性应该为一个多值属性，如图 3.27(a)所示，将该多值属性分离之后，将生成一个新的实体 StudentHonor，该实体是一个弱实体，它与原来实体 Student 之间存在一个一对多联系，如图 3.27(b)所示。

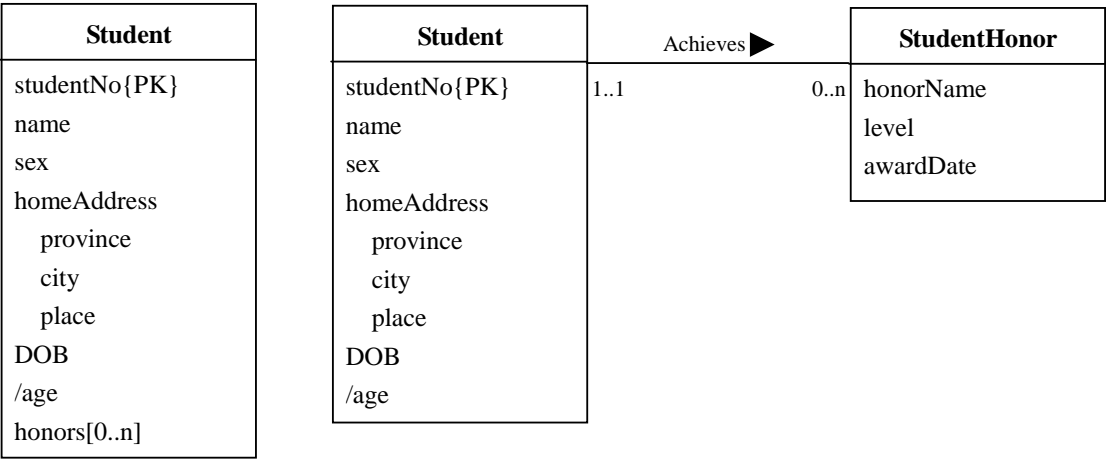


图 3.27 (a) Student 实体

图 3.27 (b) 多值属性的分离

概念设计得到的 ER 模型经过以上四个步骤的进一步转化和调整,剩下的元素类型已经比较单一,不存在多对多联系、复杂联系以及多值属性,只有实体、一对多二元联系、一对一二元联系和一对一递归联系,这些元素向关系模型的映射相对比较简单,下面开始逐一把这些 ER 模型中的元素映射为关系模型中的关系表示。

5) 强实体

对于数据模型中的每个强实体,创建一个关系(表)包含该强实体的所有简单属性,对于复合属性,应根据需要将复合属性拆分为多个简单属性。同时,标识关系的主关键字。例如,图 3.27(b)中的学生实体 Student 为强实体,为其创建一个关系,并将其中的复合属性 homeAddress 根据需要拆分为三个简单属性。结果如下:

Student(studentNo, name, sex, enrollmentYear, province, city, place, DOB, age)

主关键字 PK(studentNo)

6) 弱实体

对于数据模型中的每个弱实体,同样创建一个关系包含该弱实体的所有简单属性,对于复合属性,应根据需要将复合属性拆分为多个简单属性。不过,弱实体的部分甚至全部主关键字属性需要从该弱实体所依赖的主实体中获取,因此,弱实体主关键字的确认需要等到它所依赖的主实体以及它们之间的联系映射之后完成。例如,图 3.27(b)中的学生奖惩记录 StudentHonor 为弱实体,为其创建一个关系,但该关系主关键字暂时空缺,结果如下:

StudentHonor(honorName, level, awardDate)

主关键字 PK 暂时空缺。

7) 一对多二元联系

对于数据模型中每个一对多二元联系,联系中的“一方”实体被制定为父实体,“多方”实体被指定为子实体,将父实体主关键字属性的副本作为外关键字,放到对应子实体的关系中,同时,联系如果有属性,应将联系的属性一并放入子实体之中。例如,图 3.27(b)中学生实体 Stuent 与学生奖惩记录实体 StudentHonor 之间的联系 Achieves 为一个一对多联系,其中 Student 为“一方”实体,

StudentHonor 为“多方”实体，将 Student 实体主关键字副本放入 StudentHonor 实体中作为外关键字，同时弱实体 StudentHonor 的主关键字也可以确定，结果如下：

```
Student(studentNo, name, sex, enrollmentYear, province, city, place, DOB, age)
主关键字 PK(studentNo)

StudentHonor(honorName, level, awarddate, studentNo)
主关键字 PK(studentNo, honorName)
外关键字 FK(studentNo) 引用 Student(studentNo)
```

8) 一对一二元联系

对于数据模型中的一对一联系，同样可以参照一对多联系映射为外关键字的思想，但是仅仅通过基数约束已经没有办法区分联系中的父实体与子实体了。不过在这种情况下，联系双方实体的参与性约束可能会起到帮助作用。根据联系双方实体的参与性约束不同，一对一联系分为三种情况，下面分别予以讨论。

(1) 一对一联系中双方实体都是强制参与情况

在这种情况下，联系双方实体处于对等地位，应该合并联系双方实体为一个新的关系，并选择原实体的一个主关键字作为新关系的主关键字，原双方实体中的其它主关键字或候选关键字都作为候选关键字。新得到关系应包含联系双方实体的所有属性以及联系的属性（如果有的话）。例如，假设每个学生都有一个爱好，爱好作为一个弱实体而存在，爱好弱实体具有名称和达到水平等属性，那么学生实体与爱好实体之间是一个一对一联系，并且联系双方都是强制参与，如图 3.28 所示，这时可以将这两个实体合并为一个实体，将爱好弱实体的属性并入学生实体，得到一个关系，学生实体的主关键字仍为新实体的主关键字，结果如下：

```
Student(studentNo, name, sex, likename, likelevel)
主关键字 PK(studentNo)
```

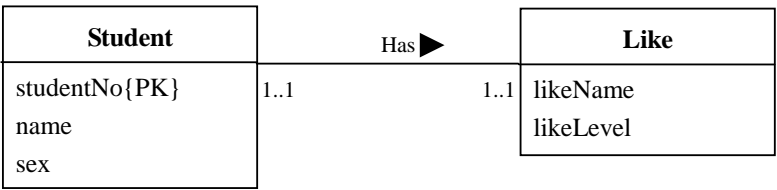


图 3.28 一对一联系双方都是强制参与

需要注意的是，以上讨论的情况是联系双方实体间不存在其它直接联系的情况，如果联系双方实体间除了存在一对一联系之外，还存在其它联系，那么其它联系的存在可能会妨碍双方实体的合并，这时可能需要采用第三种情况中的方法，采用外关键字机制表示联系，对于如何确定父子实体可以参照第三条中的讨论。

(2) 一对一联系中一方实体强制参与，另一方实体可选参与情况

在这种情况下，可以通过双方实体在联系中的参与性约束不同区分父实体和子实体。在联系中可以指定可选参与方的实体为父实体，指定强制参与方的实体为子实体，按照一对多联系情况中的外关

键字的方法，把父实体的主关键字副本作为外关键字放到子实体中。同时，如果该联系也具有属性，这些属性应该一同被放入子实体中。例如，图 3. 29 表示了教师实体 Teacher 与系实体 Department 之间的联系管理 Manages，表示每个系应该有一名系主任负责管理该系，Manages 联系是一个一对一联系，其中 Teacher 在联系中为可选参与，Department 在联系中为强制参与（即每个系必须有一名系主任），因此，对于联系 Manages，可以将教师实体 Teacher 的主关键字副本作为外关键字放入系实体 Department 中，结果如下：

Teacher(teacherNo, name, sex, DOB, idCardNo, researchfield, officeTel, officeRoom)
主关键字 PK(teacherNo)
Department(departmentNo, name, officeTel, homepage, officeRoom, **directorNo**)
主关键字 PK(departmentNo)
外关键字 FK(directorNo) 引用 Teacher(teacherNo)

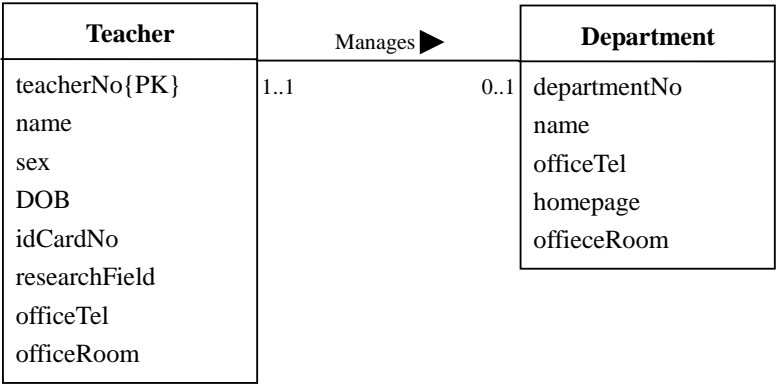


图 3.29 一对一联系中一方可选参与

（3）一对一联系中双方实体都是可选参与情况

在这中情况下，无论通过基数约束还是参与性约束，都无法确定联系双方实体中哪一方为父实体，哪一方为子实体，因此，一般情况下，可以任一指定一方实体为父实体，另一方实体为子实体。如果双方实体间还有其它联系存在，可以根据其它联系作出更好的决定。

虽然在这种情况下可以任意指定联系双方中的父实体和子实体，也就是说联系双方实体处于对等地位，但如果仔细思考，它们仍然可能存在区别。虽然联系双方都是一方实体，都是可选参与，但是参与性的强弱仍然可能不同，这种不同是 ER 模型本身所不能表达，但是在用户的使用环境下可能的确存在，它是潜在的需求，需要通过与用户沟通进行确认。一旦能够确认联系双方参与性的强与弱，它就可以为指定父实体和子实体提供了更好的选择，一般可以指定参与性较弱一方为父实体，参与性较强一方为子实体，把父实体的主关键字副本作为外关键字放到子实体中。例如，图 3. 30 中学生实体 Student 和班级实体 Class 之间的联系管理 Manages，表示每个班级应该有一名班长负责管理班级，但是允许班长暂时空缺，该联系是一个一对一联系，并且在联系中 Student 与 Class 均为可选参与，但是显然学生中只有很少一部分能够担任班长，并且大部分班级应该都有班长存在，即学生在该联系中的参与性较弱，班级在该联系中参与性较强，因此，可以选定 Student 为父实体，Class 为子实体，将父实体 Student 的主关键字副本作为外关键字放到子实体 Class 中，结果如下：

Student(studentNo, name, sex)
主关键字 PK(studentNo)
Class(classNo, totalMember, **monitor**)
主关键字 PK(classNo)
外关键字 FK(monitor) 引用 Student(studentNo)

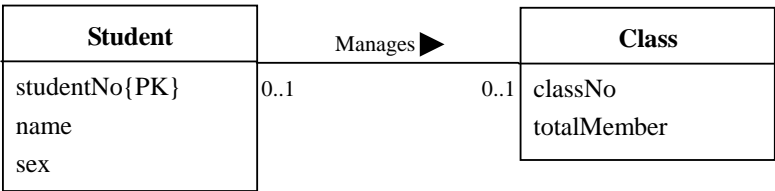


图 3.30 一对一联系中双方均为可选参与

9) 一对一递归联系

一对一递归联系其实是一对一联系的一种特殊情况，因此可以按照一对一联系的原则进行处理，只不过由于参与联系双方为同一实体，映射时稍有区别。对于联系双方实体都是强制参与的一对一递归联系，可以将该递归联系与实体表示为一个关系，在关系中增加主关键字副本作为外关键字。对于一方强制参与、一方可选参与的一对一递归联系，既可以象双方实体强制参与情况那样，创建一个含有主关键字副本作为外键的单关系，也可以创建一个新关系表示这种联系，所创建的新关系应该只有两个属性，两个属性都为主关键字的副本。对于双方都是可选参与的一对一递归联系，可以参照前面去除多对多递归联系中所述的方法，创建一个新的关系。例如，假设每个班级可能有联谊班级，但最多只能有一个联谊班级，那么班级自身之间存在一个递归联系，并且班级在该联系中的角色一方为强制参与，另一方为可选参与，因此可以将班级的主关键字一个副本作为外关键字放入本身，用一个单关系表示该联系。结果如下所示：

Class(classNo, totalMember, **friendClassNo**)
主关键字 PK(classNo)
外关键字 FK(friendClassNo) 引用 Class(classNo)

通过以上步骤，一个 ER 模型映射为关系模型就基本完成了，需要注意的是，在由 ER 模型向关系模型映射的过程中，应该形成相应的文档，记录由 ER 模型映射为关系模型过程中得到的关系模式以及支持文档，即逻辑数据模型应该由 ER 模型、关系模式以及支持文档三个部分组成。同时，在逻辑数据库设计阶段，概念数据库设计阶段得到的实体以及联系有可能会增加、减少或者发生变化，也有可能产生新的候选关键字或主关键字，因此，在逻辑数据库设计过程中或完成之后，概念数据库设计过程中得到的数据字典也可能需要更新，以保持数据字典与逻辑数据模型一致。

在以上 ER 模型向关系模型的映射完成之后，还需要对得到的逻辑数据模型进行复查、验证与确认，一般步骤如下：

1) 使用规范化的方法验证关系模式

上面提到，逻辑数据库设计得到的结果—逻辑数据模型应该由 ER 模型、关系模式以及支持文档

三个部分组成，关系模式应该是其中的核心内容和本质反映，本步骤的目的就是对逻辑数据模型中的关系模式进行验证和确认，考察该模型是否符合关系模型的基本要求。本章随后将介绍的规范化理论和技术将是验证关系模式的主要方法，规范化技术用于评价和改进关系模式，使其满足不同的约束，避免不必要的重复，并且保证数据的一致性。

2) 结合用户事务验证关系模型

该步骤的目的是确认得到的逻辑数据模型能够支持用户视图需要的事务，在概念数据库设计的第7步骤中曾经描述过这种检查，本步骤是对概念模型转化为逻辑模型之后进行确认，以保证在由概念模型得到逻辑模型过程中没有引入错误。在验证过程中，可以通过逻辑模型中的关系以及关系上的约束，尝试手工用户视图要求的事务对数据的操作，如果逻辑模型能够支持所有事务对数据的操作，该逻辑模型就得到了确认。否则，模型中就可能存在错误，这时就需要返回概念模型，对存在问题的数据模型部分进行检查，并对这部分概念模型到逻辑模型的映射过程进行检查，直至解决存在的问题。

3) 定义模型的完整性约束

该步骤的目的是定义用户视图中可能给出的完整性约束。该步骤只从逻辑模型的层次或高度定义完整性约束，从而根据用户需求对模型进行加强限制，并不考虑具体 DBMS 将如何实现。关系模型中需要考虑的约束一般有以下几种：

- (1) 空与非空约束
- (2) 属性域约束
- (3) 实体完整性（或主键）约束
- (4) 引用（或参照）完整性约束
- (5) 用户定义完整性（企业）约束

4) 与用户一起复查逻辑数据模型

本步骤的目的是与用户一起确认以上得到的局部逻辑数据模型正确描述了用户视图，在前面概念数据库设计中也有这个步骤，其作用是相同的，只有最终的逻辑数据模型得到用户的确认，才能说明它是符合要求的，进而才能进行进一步设计工作。

如果系统中只有一个用户视图，逻辑数据库设计到此就完成了，如果系统中存在多个用户视图，还需要额外的一个步骤，就是将局部逻辑数据模型合并为表示企业所有用户视图的全局逻辑数据模型。

如果系统中存在多个用户视图，通过以上设计步骤和过程，虽然针对每个用户视图的每个局部逻辑数据模型应该是正确的，但是它可能只是企业一个或一组用户的理解和描述，也就是说，这个模型是局部的和不全面的，或者是不完整的，因为由多个用户视图得到的不同的局部逻辑模型可能存在矛盾或者重复的地方，因此，必须把这些局部的逻辑数据模型合并为全局逻辑数据模型，在合并过程中，需要全力解决视图之间可能的冲突和存在的重复，同时还需要对新模型进行验证和复查。

对于一个简单的数据库设计，可能只有少量的视图，并且每个视图中实体和联系较少，因此比较各局部模型，将它们合并到一起解决存在的差异和冲突可能并不是十分复杂，然而对于一个较复杂的系统，必须使用系统化的方法进行合并。将局部逻辑数据模型合并为全局模型过程中的主要任务包括：

复查/关系的名称和内容以及它们的候选关键字，复查联系/外关键字的名称和内容，合并局部数据模型中的实体/关系和外关键字，检查完整性约束，画出全局 ER/关系图，更新相应文档。模型合并工作与设计者对系统的整体理解以及经验有很大关系，具体不再论述。

对于本书采用的 Teaching 数据库系统案例，上一小节得到的概念数据库设计结果经过以上步骤得到的全局关系图如图 3.31 所示。

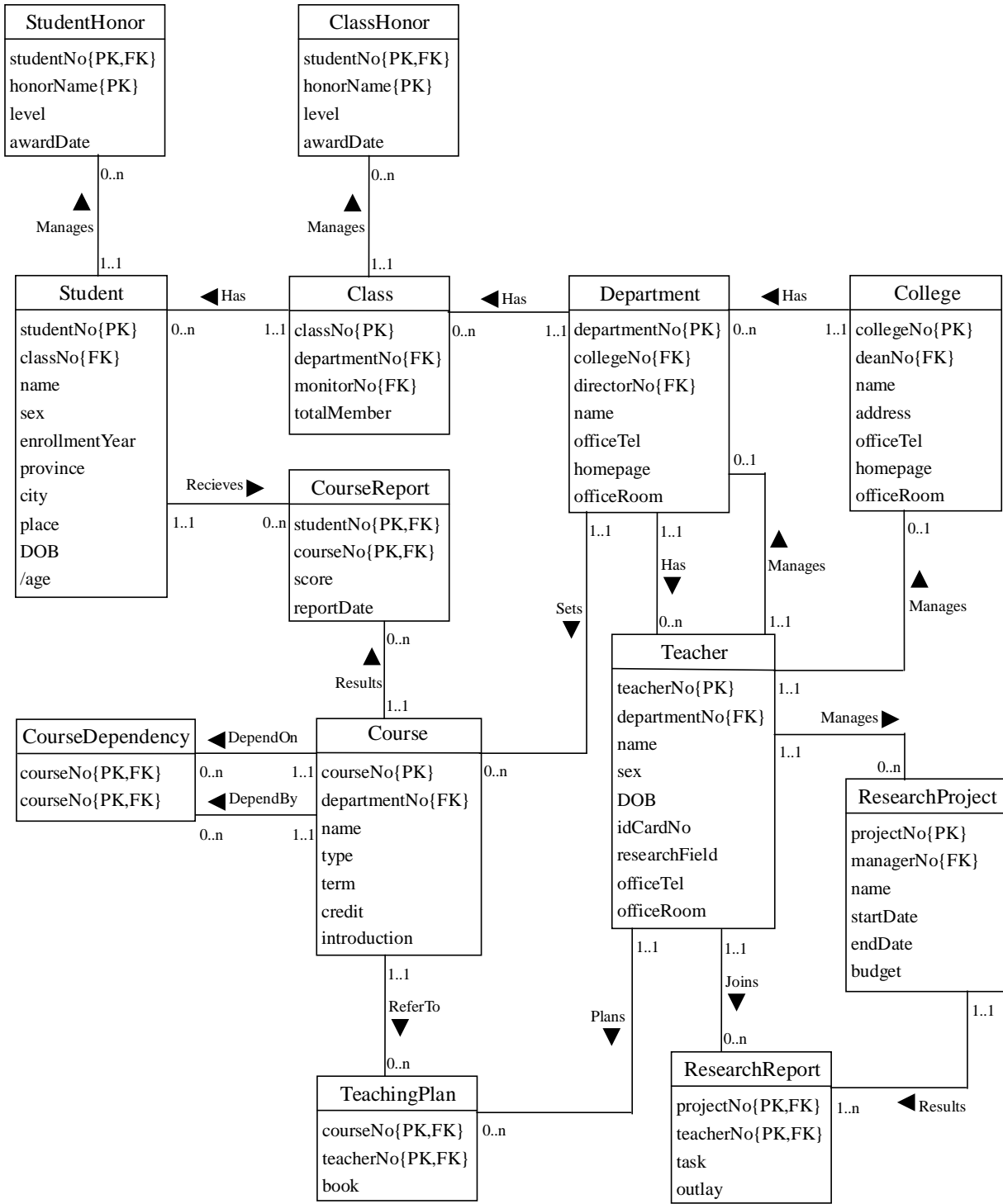


图 3.31 Teaching 案例的全局关系图

在逻辑数据库设计完成之后，数据库设计下一个阶段是物理设计。物理设计是描述逻辑数据库设计结果如何在具体目标 DBMS 上实现的过程，这些描述包括基本关系、文件组织、为实现数据高效访问而建立的索引，以及完整性约束、安全策略等。

逻辑数据库设计关注于“是什么”，物理数据库设计则关注于“如何做”，它们需要不同的技能。逻辑数据库设计要求设计者清楚常用的关系模型，并能够将由用户需求抽象出的概念模型映射为关系模型中的元素表示，而物理设计则要求设计者对目标 DBMS 的功能以及特性具有充分的了解，让逻辑数据库设计阶段得到的逻辑模型能够最终在目标 DBMS 上高效实现。

物理数据库设计并不是一项孤立的活動，该阶段的设计通常会反馈于逻辑设计以及应用程序之间。例如，在物理数据库设计阶段，为了提高系统性能可能会做出一些取舍，像合并关系，人为引入一定数据冗余，而这些活动可能会影响到逻辑模型的结构，也会影响到应用程序的设计。

由于不同数据库管理系统产品所提供的物理环境、存取方法以及存储结构有很大差别，供设计人员使用的设计变量、参数选取范围也有很大不同，因此，本书将不对物理数据库设计进行论述。

3.4 关系模式规范化



内容提要

通过上一小节内容，我们将根据用户需求而创建的 ER 模型映射为关系模型，关系模型是当前绝大部分 DBMS 所支持的数据模型，也就是说，我们的在逻辑数据库设计阶段得到的逻辑数据模型将能够被当前绝大部分 DBMS 所支持，并能够最终在 DBMS 中实现。在逻辑数据库设计阶段我们得到的结果是一组关系模式以及关系模式之上的约束，在它们最终被 DBMS 实现之前，我们有必要对这些关系模式进行进一步考察，分析它们是否仍然存在问题或者某些不好的特征，从而影响最终在 DBMS 中实现的效果，这是本节将要讨论的主要内容。本节主要包含如下内容：

- 问题的提出；
- 函数依赖；
- 1NF，2NF，3NF，BCNF 的概念和特点；
- 关系模型规范化过程
- 关系数据库设计中反规范化的思想。

3.4.1 问题的提出

通过概念数据库设计和逻辑数据库设计我们得到了一组关系模型，这些关系模式对于关系模型来讲是否具有良好的性质，到目前为止我们并没有涉及。在逻辑数据库设计中，我们是通过给出的一系列方法和过程，从而把概念数据库设计得到的 ER 模型转化为一组关系模式，如果按照不同的方法和过程，可能会得到不同的关系模式集合，那么，通过不同方法得到的不同的关系模式集合是否都是好的关系模式呢？或者说，针对一个具体的概念数据库设计结果，应该如何构造一个适合于它的关系模式集合？即应该构造几个关系模式，每个关系模式应该由哪些属性组成。另外，对于一个给定的关系模式，它是否具备良好的性质，有否一个评价标准？这些都是关系模式规范化中将要讨论的内容，在详细讨论之前，首先分析一个具体例子，从中或许能够得到一些启发。

假设由概念数据库设计结果，通过逻辑数据库设计，我们得到逻辑模型中的一个关系模式 TeacherCourse 如图 3.32 所示。

teacherNo	teacherName	officeRoom	courseNo	courseName	credit
T001	王波	诚字楼 A601	C001	信息系统	2
T001	王波	诚字楼 A601	C002	数据结构	3
T001	王波	诚字楼 A601	C003	数据库系统	3
T002	黄涛	诚字楼 A602	C001	信息系统	2
T002	黄涛	诚字楼 A602	C004	操作系统	3
T003	耿小亮	诚字楼 A603	C005	组成原理	3

图 3.32 TeacherCourse 关系模式

其中，关系模式中的各属性 teacherNo、teacherName、officeRoom、courseNo、credit 分别表示教师编号、教师姓名、教师办公室、教师所讲授课程编号、课程名称和课程学分。通过对以上关系模式进行初步考察，可以发现其中存在以下问题：

1) 数据冗余

从图 3.32 很容易看出关系 TeacherCourse 中存在大量数据冗余。当一名教师讲授多门课程时，需要重复记录该名教师的相关信息。在图 3.32 中，教师“王波”讲授了 3 门课程，因此，王波教师的相关信息被存储了 3 遍。当多名教师讲授同一门课程时，同样需要重复记录该门课程的相关信息。在图 3.32 中，教师王波与教师黄涛都讲授“信息系统”这门课程，因此，信息系统课程的相关信息被存储了 2 遍。

2) 数据更新异常

存在数据冗余的关系可能会带来数据更新异常的问题，数据更新异常分为插入异常、删除异常和修改异常。数据更新异常是一种不好的现象，在数据库设计中应该尽量避免。

对于图 3.32 的例子，由于关系模式 TeacherCourse 的主关键字为教师号 teacherNo 和课程号 courseNo 的组合，即 teacherNo 和 courseNo 属性都不能为空，因此，当向关系 TeacherCourse 中插入一名新教师时，还必须同时至少插入一门该名教师所讲授课程的信息，如果该门课程的信息在关系中已经存在，那么新插入的该门课程信息必须与原有该门课程信息一致。另外，如果新插入的教师当

前没有讲授任何课程，那么该名教师将无法插入到该关系中。对于新插入一门课程的情况存在同样的问题。

以上针对关系模式 TeacherCourse 的插入异常进行了讨论，除此之外，根据以上分析方法，很容易发现它还存在删除异常和修改异常，在此不再详细讨论。

由于例子中给出的关系模式 TeacherCourse 存在以上不好的现象，因此，可以说它不是一个好的关系模式，那么我们能否尝试将该关系模式改造一下，让它能够变得好一些，或者让他能消除以上存在的部分问题呢？其实把以上关系模式改造的更好一些并不是十分困难，至少可以尝试一下。假设我们把关系模式 TeacherCourse 改造为两个关系模式 Teacher 与 Course，需要注意的是，将 TeacherCourse 拆分为两个关系模式 Teacher 与 Course 之后，教师讲授课程的联系不能丢失，因此，需要将教师编号放入关系模式 Course 中作为外关键字，如图 3.33(a) 和 (b) 所示。

teacherNo	teacherName	officeRoom
T001	王波	诚字楼 A601
T002	黄涛	诚字楼 A602
T003	耿小亮	诚字楼 A603

图 3.33 (a) Teacher 关系模式

courseNo	courseName	credit	teacherNo
C001	信息系统	2	T001
C002	数据结构	3	T001
C003	数据库系统	3	T001
C001	信息系统	2	T002
C004	操作系统	3	T002
C005	组成原理	3	T003

图 3.33 (b) Course 关系模式

同样可以考察一下，经过改造之后得到的两个关系模式 Teacher 与 Course 是否比原来关系模式有所好转呢？很容易看出，改造后的两个关系模式数据冗余明显减少了。在改造之前的关系模式 TeacherCourse 中，如果一名教师讲授多门课程，那么该名教师的姓名和地址需要保存多次，而改造之后，同一教师的姓名与地址只需要保存一次就可以了，该数据冗余的减少同样避免了部分数据更新异常的问题。

在以上分析中，通过把关系模式 TeacherCourse 分解为两个关系模式 Teacher 与 Course，从而消除了原来关系模式中存在一些不好的现象，那么分解之后得到的关系模式 Teacher 与 Course 是否还存在问题呢？对于 Course 关系模式，如果同一门课程可以有多个教师讲授，将会出现什么情况？它是否也存在上面提到的数据冗余和数据更新等不好的现象？答案显然是肯定，也就是说 Course 关系模式仍然需要改造。

以上我们是通过一个具体的例子进行的考察，而我们需要得到一个一般性的结论，因此需要进一步思考，对于给定的任何一个一般的关系模式，我们应该如何判断其是否存在的问题，如果存在问题，

又该如何改造？

如果对上面例子中给出的关系模式 TeacherCourse 进一步考察，应该可以发现该关系模式的属性之间并不是孤立的，而是有联系的。首先，根据实际隐含的语义，该关系模式的候选关键字只有一个，应该为属性 teacherNo 与 courseNo 的组合，因此，(teacherNo, courseNo) 被选定为主关键字，再考察其它属性，例如，teacherName 属性，该属性其实并不需要主关键字两个属性确定，而只需要主关键字属性中的 teacherNo 一个属性就可以确定，同样 officeRoom 属性也只需要 teacherNo 属性就可以确定。在前面对关系模式 TeacherCourse 的分析中，恰恰是这两个属性存在数据冗余，同样，在改造后的关系模式 Course 中，如果每门课程可以有多个教师讲授，该关系模式的主关键字应该为 (courseNo, teacherNo)，而课程名 courseName 只需要主关键字属性中的 courseNo 一个属性就可以确定，而 courseName 属性在 Course 关系中也存在数据冗余。通过以上对关系模式 TeacherCourse 与 Course 的分析，初步发现关系模式中导致数据冗余的属性具有相似之处，它们都可以由主关键字的部分属性就可以确定，这并不是偶然的巧合，而是必然。

通过以上具体例子的考察和分析，我们可以得到两个一般性的结论：

- (1) 根据语义，关系模式属性之间存在联系；
- (2) 关系模式属性之间不好的联系可能会导致问题。

下面我们将从关系模型属性之间的联系开始，讨论一般关系模式可能存在的问题以及解决方法。由于关系模型是建立在严格的数学基础之上，因此，先从数学上的函数依赖开始讨论。

3.4.2 函数依赖

1) 数据依赖

在前面概念数据库设计的讨论中，我们知道现实世界中实体与实体之间普遍存在着联系，不仅如此，同一实体的属性之间也可能存在联系，那么对应到逻辑数据库设计中，同一关系模式中的属性之间同样可能存在联系。我们把同一关系模式中属性之间相互依存或者相互限制的关系称为属性之间的**数据依赖 (Data Dependency)**，数据依赖是现实世界和关系模式中属性之间相互联系的抽象，是数据内在的性质。另外，数据依赖是语义的体现，某些属性之间存在数据依赖是由它们之间的实际语义决定的。例如，在学生关系模式 Student(studentNo, name, sex, age) 中，属性学号 studentNo 与姓名 name 之间应该存在某种联系，学生学号确定了，姓名也随之确定，也就是说学号与姓名这两个属性之间存在数据依赖，即学号能够决定姓名，这一数据依赖存在的语义基础是学生的学号是唯一的，如果没有学号唯一这个语义基础，学号决定姓名这一数据依赖也就不可能存在。

数据依赖的形式和类型是多样的，到目前为止，人们已经提出了多种类型的数据依赖，其中在数据库设计中能够使用到的有函数依赖 (Functional Dependency, FD)、多值依赖 (Multi-Valued Dependency) 和连接依赖 (Join Dependency)，本节将仅仅针对其中最常用的也是最重要的函数依赖进行讨论。

2) 函数依赖

函数依赖是数据依赖的一种，它描述了一个关系模式中属性或属性组之间这样一种联系，假设 A 和 B 均为关系模式 R 的属性或属性组，如果对于 A 中的每个值，B 中都有唯一一个值与之对应，那么

就称 B 函数依赖于 A ，或 A 函数决定 B ，记作 $A \rightarrow B$ 。其中 A 称作决定方 (Determinant)。如果 $A \rightarrow B$ 的同时 $B \rightarrow A$ ，那么 A 和 B 相互决定，记作 $A \leftrightarrow B$ 。

根据定义，函数依赖可分为平凡的函数依赖 (Trivial Functional Dependency) 和非平凡的函数依赖 (Nontrivial Functional Dependency)。在上面讨论的关系模式 R 中，如果 $A \rightarrow B$ ，并且 $B \subseteq A$ ，则函数依赖 $A \rightarrow B$ 为平凡的函数依赖；若 $A \rightarrow B$ ，并且 $B \not\subseteq A$ ，则函数依赖 $A \rightarrow B$ 为非平凡的函数依赖。显然，平凡的函数依赖是指不可能不满足的函数依赖，它并不包含语义的抽象，讨论它也没有很大意义，因此在后面的叙述中，如果没有特别声明，总是指非平凡的函数依赖。

由于函数依赖是数据依赖的一种，因此属性之间的函数依赖也是关系模式中属性之间语义特性的反映。例如，在前面学生关系模式 $Student(studentNo, name, sex, age)$ 的例子中， $studentNo$ 与 $name$ 、 sex 以及 age 属性之间的数据依赖就是一种函数依赖，因为对于每个 $studentNo$ 取值， $name$ 、 sex 以及 age 都有唯一值与之对应，其中 $studentNo$ 是决定方。

在讨论了关系模式中属性之间的函数依赖之后，对于一个关系模式的表示又增加了新的内容，不仅要表示关系模式中各个属性以及属性域，还要表示各属性之间的数据依赖，主要是函数依赖。本节主要针对关系模式中属性之间的联系展开讨论，而不涉及属性域问题，因此为了方便讨论，在本节我们可以用 $R(U, F)$ 表示一个关系模式，其中 R 表示关系名， U 表示关系模式的属性集合， F 表示关系模式中属性之间存在的数据依赖集合。

一般情况下，对于属性或属性组 X 和 Y ，如果 X 和 Y 之间存在一对一 (1:1) 的映射，则 X 和 Y 是相互决定，即 $X \leftrightarrow Y$ ；如果 X 和 Y 之间存在一对多 (1:*) 的映射，则 X 和 Y 之间存在函数依赖 $Y \rightarrow X$ ；如果 X 和 Y 之间存在多对多 (*:*) 的映射，则 X 和 Y 之间不存在函数依赖关系。图 3.34 的 (a)、(b) 和 (c) 分别表示了以上三种情况。

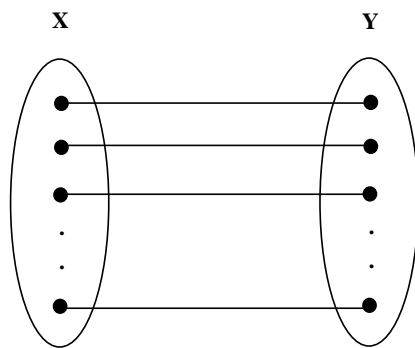


图 3.34 (a) X 与 Y 之间的一对一映射

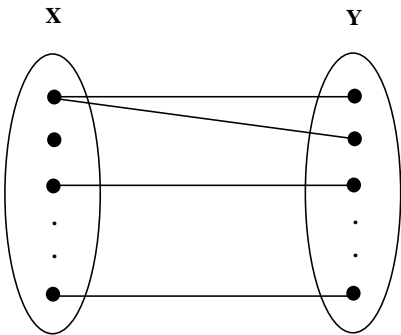


图 3.34 (b) X 与 Y 之间的一对多映射

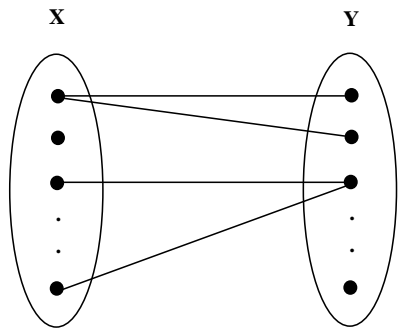


图 3.34 (c) X 与 Y 之间的多对多映射

需要注意的是，函数依赖不是指关系模式 R 的某个或某些关系满足的约束条件，而是指 R 的一切关系都必须满足的约束条件。因此，在标识一个关系模式中属性之间的函数依赖时，必须区分一个属性在某一时刻具有的不同属性值，以及该属性在任意时刻可能具有的所有属性值的集合，这对于函数依赖的确定是十分重要的。

例如，对于学生关系 Student(studentNo, name, sex, age) 中，考虑属性 studentNo 和 name 两个属性，前面已经讨论过，对于每个 studentNo 取值，name 有唯一值与之对应，因此存在函数依赖 studentNo→name，如果考察该关系模式中当前的每个关系，也可能会发现，对于每个 name 也可以确定一个 studentNo，那么现在是否能得出结论 name→studentNo 呢？如果我们考察的关系表示了该关系模式在 studentNo 和 name 属性上所有可能的取值，那么 name→studentNo 是成立的，然而事实可能并不是这样，我们考察的关系可能只是该关系模式某一时刻的值，而这并不是标识函数依赖时所关注的，函数依赖关注的是该关系模式所有可能取值的情况，因此，不能够根据当前关系取值确定函数依赖 name→studentNo。为了能够准确标识关系模式中属性之间的函数依赖，应该清楚理解每个属性的目的。显然，studentNo 属性的目的是能够唯一标识每个学生，一旦知道某个学生的学号，该学生的其它属性就随之确定，而 name 属性表示学生的姓名，在当前考察的关系中或者大部分情况下，学生姓名不会出现重复，但是从该属性所有取值情况看，学生姓名可能会出现重复的现象，虽然很少出现，因此，name 并不能够确定 studentNo 和其它属性，即不能确定 name→studentNo。在考虑该关系模式所有可能取值情况下，函数依赖 studentNo→name 依然是成立的。

关系模式属性间的函数依赖的确定，对于确定该关系模式的候选关键字具有重要帮助，能够决定

关系模式中所有属性的属性或属性组合即可作为关系模式的候选关键字。

3.4.3 规范化的概念

规范化 (Normalization) 是一种形式化技术, 经常被用来对关系模式进行测试, 以验证它是否满足或者违反了某个给定的范式要求, 从而具有某些不好的性质, 并据此对关系模式进行分解, 直至满足规范化的要求。规范化技术和过程最初是由 E. F. Codd 提出的, 最早提出了三个范式, 即第一范式 (the First Normal Form, 1NF)、第二范式 (the Second Normal Form, 2NF) 和第三范式 (the Third Normal Form, 3NF)。后来, R. Boyce 和 E. F. Codd 又提出了一种增强的第三范式, 称为 Boyce-Codd 范式 (Boyce-Codd Normal Form, BCNF)。以上所有这些范式都是基于关系模式中属性之间的函数依赖进行分析和讨论的。后来, 其它人等又提出了第四范式 4NF 和第五范式 5NF, 它们对关系的要求更为严格, 但是一般在数据库设计中, 用到 4NF 和 5NF 的情况非常少, 因此本节将仅仅针对在数据库设计中经常用的 1NF 到 BCNF 进行论述, 而不涉及 4NF 和 5NF。

在下面范式的定义中, 可能会用到主属性和非主属性的概念, 在此首先给出这两个概念的确切定义。在关系模式中我们曾经给出过候选码和主码的定义, 在当时的定义中, 候选码是能够唯一标识关系中每个元组的一个或一组最小属性的集合。一个关系中也可能会有多个候选码或称为码, 可以选定其中一个为主码。在此定义, 包含在任何一个候选码中的属性, 都称为**主属性 (Prime Attribute)**, 不包含在任何候选码中的属性称为**非主属性 (Nonprime Attribute)** 或非码属性。

3.4.4 第一范式 (1NF)

1NF 是所有关系数据库中所有关系模式都应该满足的最低要求, 它要求关系模式的所有属性都是不可分的基本数据项, 即关系的每一行和每一列的相交部分 (称为格) 只能有一个值。满足 1NF 要求的关系或表称为规范化的关系或表, 不满足 1NF 要求的关系或表称为非规范化的关系或表。一个关系模式 R 满足 1NF 可以记作 $R \in 1NF$ 。需要注意的是, 1NF 是关系数据库中所有关系模式都必须满足的要求, 而随后将介绍的其它范式的要求都是可选的。

为了将一个非规范化的表转化为满足 1NF 要求的表, 需要消除表中一格包含多个值的情况, 通常有三种做法:

1) 增加行的方式。

对于关系中每行可能含有多个数据值的数据项, 通过增加行的方式把该数据项中的多个值放入不同行中, 从而使得该数据项在每行只能有一个值存在, 同时对于增加行中的其它数据项需要填充重复数据。通过这种方法虽然能够将一个非规范化的关系转化为符合 1NF 的关系, 但是同时它在关系中引入了大量数据冗余, 这些冗余将在后面的规范化过程中消除。另外, 增加行的方式可能会导致原关系的主码地位发生变化。

2) 增加列的方式

对于关系中每行可能含有多个数据值的数据项, 通过增加列的方式把该数据项中的多个值放入不同列中, 通过将该数据项拆分为多个列, 使得拆分后的每个列对于关系的每行只能有一个值存在。该方法在将非规范化关系改造为 1NF 的同时, 不会引入大量数据冗余, 但是这种方法的使用具有明显的局限性, 对于关系中每行可能含有多个数据值的数据项, 只有能够确定该数据项最多可能含有几个值

的情况下，采能够使用，否则将无法确定应该增加多少列来消除该数据项的多值问题。

3) 分解表的方式

对于关系中每行可能含有多个数据值的数据项，通过将该数据项移到另一个独立的新关系中，同时将原关系中的主关键字属性也复制到这个新的关系中，从而保证原关系中属性之间的联系，然后重新确定新关系的主关键字。该方法将原关系拆分为两个或者多个关系，从而使得拆分后的每个关系都符合 1NF。另外，该方法在使关系符合 1NF 的同时没有带来数据冗余问题，但是该方法将原来的一个关系分解为两个和多个关系，对于数据查询的效率可能会有影响。

下面通过一个例子来说明如何把一个非规范化的关系转化为满足 1NF 要求的关系。图 3.35 显示了一个关系 Student，其中 phoneNo 属性每行和每列相交的小格可能有 2 个值，即每个学生可能具有两个电话号码。

studentNo	name	sex	phoneNo
S001	张明	男	88451234 13576342741
S002	赵强	男	88451236
S003	王红	女	88452315 13019124761

图 3.35 Student 关系

该关系显然不满足 1NF 的要求，为了将该关系改造为满足 1NF 要求，按照上面提到的第一种方法一增加行的方式，改造结果如图 3.36 所示：

studentNo	name	sex	phoneNo
S001	张明	男	88451234
S001	张明	男	13576342741
S002	赵强	男	88451236
S003	王红	女	88452315
S003	王红	女	13019124761

图 3.36 增加行方式改造后的 Student 关系

该方式通过为关系中 phoneNo 属性在每行可能的两个取值增加一行，从而限制每行中 phoneNo 属性只能取一个值，改造之后的关系模式满足了 1NF 的要求。需要注意的是，通过增加行的方式改造后的关系模式的主码已经发生了变化，在原来关系模式中，主码为学号 studentNo，而改造之后的关系模式中，主码变为了学号 studentNo 与电话号码 phoneNo 的组合（studentNo，phoneNo）。另外，显而易见，使用这种改造方法的结果增加了关系中的数据冗余，这些冗余后续的规范化过程中将需要进一步消除。

如果按照第二种改造方法一增加列的方式对 Student 关系进行改造，改造结果如图 3.37 所示：

studentNo	name	sex	phoneNo	mobilephoneNo
S001	张明	男	88451234	13576342741
S002	赵强	男	88451236	
S003	王红	女	88452315	13019124761

图 3.37 增加列方式改造后的 Student 关系

该方式根据关系中 phoneNo 属性每行可能取两个值，将 phoneNo 属性分解为两个属性，从而限定分解之后的每个属性每行只能取一个值，使得改造之后的关系模式满足 1NF 的要求。该方式改造后的关系中保持了原来关系中主码的地位，但是由于并不是每个学生都有两个电话号码，因此，mobilephoneNo 属性可能会有部分空值存在，从而造成一定存储空间浪费。另外，该方法的局限性显而易见，当前是在确定了每个学生最多具有 2 个电话号码的情况下进行的改造，如果不能确定每个学生最多能有几个电话号码，那么将 phoneNo 属性分解为几个属性呢？这一问题将无法确定。因此，该方法只有在确定属性最多取值个数情况下才能采用。

如果按照第三种方法一分解表的方式对 Student 关系进行改造，改造结果如图 3.38 所示：

studentNo	name	sex
S001	张明	男
S002	赵强	男
S003	王红	女

studentNo	phoneNo
S001	88451234
S001	13576342741
S002	88451236
S003	88452315
S003	13019124761

图 3.38 分解表方式改造后的 Student 关系

该方法通过把重复数据 phoneNo 移到一个新的关系中，同时将原来的主关键字属性 sNo 复制到新的关系中，从而消除原来表中的数据重复问题。原关系通过该方法改造将得到两个新的关系，其中第一个关系的主码仍然为 studentNo，第二个关系的主码为 (studentNo, phoneNo)，这两个新的关系都满足 1NF 的要求，同时也没有引入数据冗余，对本例来讲是一个较好的选择。

在以上三种改造方法中，第二种方法一增加列的方式具有一定的局限性，在能够确定某个属性在每行最多取值个数的情况下，不失为一种好的选择，否则就只能采用第一种方法或第三种方法。其实第一中方法和第三种方法改造的结果虽然不同，但是随着进一步的规范化，它们的结果是一样的。对于本例来将，第一种方法改造后的结果经过 2NF 的改造将得到与第三种改造方法相同的结果。在阅读了 2NF 的内容之后可以继续思考。

3.4.5 第二范式（2NF）

满足 1NF 要求的关系模式仍然可能存在问题，例如，上面通过第一种方法改造后的 Student 关系，如图 3.36 所示，它存在大量的数据冗余，而数据冗余是一种不好的现象，应该予以消除。另外，数据冗余的存在同时可能会导致了插入、删除以及更新异常等问题。为了消除这样的数据冗余以及它所带来的一系列问题，引出了第二范式的概念。

第二范式的定义是基于完全函数依赖和部分函数依赖的概念，因此，下面首先介绍完全函数依赖和部分函数依赖。

1) 完全函数依赖与部分函数依赖

假设 X 和 Y 分别为关系模式 R 的属性或属性组，如果 Y 函数依赖于 X ，但不函数依赖于 X 的任何一个真子集，那么称 Y 完全函数依赖于 X ，否则，称 Y 为部分函数依赖于 X 。或者说，对于函数依赖 $X \rightarrow Y$ ，如果移除 X 中的任一属性都将使得这种函数依赖不再成立，那么 $X \rightarrow Y$ 就是一个完全函数依赖。如果移除 X 中的某些属性，这种函数依赖仍然成立，那么 $X \rightarrow Y$ 就是一个部分函数依赖。

2) 2NF 的定义

对于关系模式 R 满足 1NF ($R \in 1NF$)，如果 R 中的任意一个非主属性都完全函数依赖于码，则称关系模式 R 满足 2NF 的要求，记作 $R \in 2NF$ 。

由定义可知，2NF 是基于部分函数依赖的概念，如果一个关系的所有码都是由单个属性构成，也就不会存在非主属性对码的部分依赖问题，那么该关系将至少满足 2NF，因此，2NF 主要针对那些具有复合码的关系。

不满足 2NF 的关系仍然存在 3.4.1 节中曾经讨论过的诸多问题，因此，需要将它继续改造，使其满足 2NF 的要求。将一个 1NF 关系规范化为 2NF 关系的主要是消除其中的部分函数依赖，消除方法是把部分依赖于码的非主属性移到一个新的关系中，同时将它们所依赖的主属性复制到这个新的关系中。

例如，对于图 3.36 所示的关系 $Student$ ，该关系的码为 $(studentNo, phoneNo)$ ，而非主属性 $name$ 和 sex 是部分函数依赖于码，它仅仅函数依赖于码的一个属性 $studentNo$ ，因此关系 $Student$ 不满足 2NF。为了将该关系改造为满足 2NF 要求的关系，将部分依赖于码的属性连同它们所依赖的属性分离出来，组成一个新的关系，从而将原关系分解为两个关系，结果与图 3.38 所示结果相同，这也说明了上面讨论的针对不满足 1NF 的关系模式的三种改造方法，经过进一步规范化，结果是相同的。

再看一个例子，设关系模式 $CourseReport(U, F)$ 中， $U = \{studentNo, courseNo, courseName, score, credit\}$ ， $F = \{(studentNo, courseNo) \rightarrow score, courseNo \rightarrow courseName, courseNo \rightarrow credit\}$ ，某时刻的一个关系如图 3.39 所示。

studentNo	courseNo	courseName	credit	score
S001	C001	计算机系统导论	2	90
S001	C002	信息系统导论	1	85
S002	C001	计算机系统导论	2	85
S002	C003	数据结构	3	80
S003	C001	计算机系统导论	2	80
S004	C002	信息系统导论	1	90

图 3.39 CourseReport 关系

该关系模式的码为 $(studentNo, courseNo)$ ，其中非主属性 $courseName$ 与 $credit$ 部分函数依赖于码，它仅仅函数依赖于 $courseNo$ 属性，因此该关系模式不满足 2NF。对该关系模式的改造方法，

可以将对码部分函数依赖的非主属性 `courseName` 与 `credit` 分离出来，放入一个新的关系中，同时将 `courseNo` 属性复制到新关系中，将新关系命名为 `Course`，它具有属性 `courseNo`，`courseName` 和 `credit`，该关系的码为 `courseNo`。原关系 `CourseReport` 经过改造之后具有属性 `studentNo`，`courseNo` 和 `score`，将其重新命名为 `CourseScore`，其码仍为 (`studentNo`，`courseNo`)。改造后得到的这两个关系模式均满足 2NF 的要求。如图 3.40 (a) 和 (b) 所示。

courseNo	courseName	credit
C001	计算机系统导论	2
C002	信息系统导论	1
C003	数据结构	3

图 3.40 (a) Course 关系

studentNo	courseNo	score
S001	C001	90
S001	C002	85
S002	C001	85
S002	C003	80
S003	C001	80
S004	C002	90

图 3.40 (b) CourseScore 关系

3.4.6 第三范式 (3NF)

尽管满足 2NF 的关系模式比满足 1NF 的关系模式能够具有更小的数据冗余，但冗余仍然存在，因而由于冗余带来的数据更新异常问题也同时存在。例如，对于关系模式 `StudentDepartment` (`U`, `F`)，其中 `U` = {`studentNo`, `studentName`, `departmentNo`, `departmentName`, `officeRoom`}，`F` = { `studentNo`→ `studentName`, `studentNo`→`departmentNo`, `departmentNo`→`departmentName`, `departmentNo`→`officeRoom`}，某一时刻该关系模式的一个关系如图 3.41 所示。显然该关系模式的码为 `studentNo`，并且通过考察该关系模式满足 2NF，但是它仍然存在问题。例如，每个系都有多名学生，在关系 `StudentDepartment` 中，每个系的相关信息需要保存多遍，这将带来很大的数据冗余。

studentNo	studentName	departmentNo	departmentName	officeRoom
S001	张明	RJ01	软件工程	A601
S002	赵强	RJ02	微电子	A501
S003	王红	RJ03	电子服务	A301
S004	李斌	RJ01	软件工程	A601

图 3.41 StudentDepartment 关系

为了说明这一问题存在的原因，并且最终解决这一问题，我们将用到传递依赖的概念。下面首先给出传递依赖的定义。

1) 传递依赖

假设 `X`、`Y` 和 `Z` 分别为关系模式 `R` 的属性或属性组，如果存在 `X`→`Y`，`Y`↛`X`，并且 `Y`→`Z` (`Z`↛`Y`)，那么称 `Z` 通过 `Y` 传递依赖于 `X`。其中条件 `Y`↛`X` 很重要，因为，如果 `Y`→`X`，则 `X`↔`Y`，那么 `Z` 将直接依赖于 `X` 而不是传递依赖。

在给出传递依赖的概念之后，再分析图 3.41 给出的 `StudentCourse` 关系的例子，可以看出其中属性 `departmentName` 和 `officeRoom` 对该关系的主属性 `studentNo` 就是传递依赖，而导致该关系模式

存在问题的原因正是这种传递依赖的存在，因此，需要对该关系模式进一步改造，消除这种传递依赖的影响，这就是第三范式的内容。

2) 3NF 的定义

关系模式 R 满足 2NF ($R \in 2NF$)，如果 R 中的任意一个非主属性都不传递依赖于码，则称关系模式 R 满足 3NF 的要求，记作 $R \in 3NF$ 。

由 3NF 的定义可知，将 2NF 关系规范化到 3NF 关系需要进一步消除关系中存在的传递依赖。消除传递依赖可以采用和消除部分函数依赖相同的方法，将对码存在传递依赖的属性移到一个新的关系中，并将这些属性的决定方也复制到新的关系中。例如，对于图 3.41 中的 StudentDepartment 关系，其中非主属性 departmentName 和 officeRoom 对主属性 studentNo 存在传递依赖，因此，该关系模式不满足 3NF。为了消除该传递依赖的存在，可以将该关系模式改造为 Student 和 Department 两个关系模式，如图 3.42(a) 和 (b) 所示。

studentNo	studentName	departmentNo
S001	张明	RJ01
S002	赵强	RJ02
S003	王红	RJ03
S004	李斌	RJ01

图 3.42 (a) Student 关系

departmentNo	departmentName	officeRoom
RJ01	软件工程	A601
RJ02	微电子	A501
RJ03	电子服务	A301

图 3.42 (b) Department 关系

3.4.7 BC 范式 (BCNF)

通过前面的讨论得知，部分依赖和传递依赖的存在是导致数据冗余以及数据修改异常的主要原因，2NF 和 3NF 通过不允许非主属性对码的部分依赖和传递依赖，从而消除了大部分数据冗余和数据更新异常问题。一般情况下，如果数据库中所有关系模式都满足了 3NF，我们可以认为该结果是良好的。然而到目前为止，我们并没有考虑主属性对码可能存在的部分依赖和传递依赖，而主属性对码可能存在的部分依赖和传递依赖同样会导致问题产生。例如，对于关系模式 CourseTimeTable (teacherNo, courseNo, prelectionDate, prelectionTime, roomNo)，各属性分别表示教师编号、课程编号、上课日期、上课时间、上课教室编号，表示教师在某个时间某个教室讲授某门课程，假设教师讲授同一门课程的地点将被分配在一个固定教室，某门课程在同一时间可能在多个教室由不同教师同时上。某时刻该关系模式的一个关系如图 3.43 所示。

teacherNo	courseNo	prelectionDate	prelectionTime	roomNo
T001	C001	2-Mar-2009	8:30	教西 C106
T001	C001	9-Mar-2009	14:30	教西 C106
T002	C001	2-Mar-2009	8:30	教西 C206
T002	C001	9-Mar-2009	14:30	教西 C206
T002	C004	4-Mar-2009	8:30	教西 C106
T003	C005	5-Mar-2009	8:30	教西 C108

图 3.43 CourseTimeTable 关系

关系 CourseTimeTable 具有两个候选关键字，分别为

(teacherNo, prelectionDate, prelectionTime), (prelectionDate, prelectionTime, roomNo),

这两个候选关键字包含共同的属性 prelectionDate 和 prelectionTime，可以选择 (teacherNo, prelectionDate, prelectionTime) 作为该关系的主关键字。CourseTimeTable 关系具有下面的函数依赖：

fd1: (teacherNo, prelectionDate, prelectionTime) → courseNo, roomNo

fd2: (prelectionDate, prelectionTime, roomNo) → teacherNo, courseNo

fd3: (teacherNo, courseNo) → roomNo

其中，fd1 与 fd2 中的决定方都为关系 CourseTimeTable 的码，因此这些依赖不会带来任何问题。唯一需要讨论的函数依赖是 (teacherNo, courseNo) → roomNo (fd3)，对于 (teacherNo, courseNo) → roomNo 虽然这是一个传递依赖，但是由于 roomNo 是候选码 (prelectionDate, prelectionTime, roomNo) 中的一个属性，为主属性，并不存在非主属性对码的部分依赖与传递依赖的问题，所以该函数依赖是 3NF 所允许的，即 CourseTimeTable 满足 3NF 的要求。

虽然该关系模式满足 3NF，但它仍然存在问题，当改变某个教师讲授某门课程的教室编号时可能需要更新多条记录。例如，对于图 3.43 所示的关系，当需要改变教师 T001 讲授 C001 课程的教室编号时，就需要更新关系中的两个元组，如果只更新了一个元组的教师编号，就会导致数据库的不一致状态。

为了消除上面分析中提到的 CourseTimeTable 存在的问题，解决 3NF 存在的不足，导致了另一种要求更严格的范式的出现，即 Boyce-Codd 范式 (BCNF)。

Boyce-Codd 范式定义：关系模式 R 满足 1NF ($R \in 1NF$)，对于 R 中属性间的函数依赖，如果每个函数依赖的决定方都是码，则 $R \in BCNF$ 。

根据 BCNF 的定义，它排除了任何属性对码的部分依赖与传递依赖，弥补了 2NF 和 3NF 只限制非主属性对码的部分依赖与传递依赖，因此，BCNF 也称为扩充的和加强的第三范式。

在给出了 BCNF 的定义之后，继续考察 CourseTimeTable 关系，由于该关系上存在函数依赖 fd3 中的决定方 (teacherNo, courseNo) 并不是 CourseTimeTable 关系的码，而 BCNF 要求函数依赖的决定方必须都为码，所以该关系模式不满足 BCNF 的要求。

为了消除存在的问题，需要将该关系模式进行改造，从而使它满足 BCNF 的要求。改造方法与前

面 2NF 和 3NF 中论述的方法相同，是将关系模式中存在的部分依赖与传递依赖分离出来。例如，对于 CourseTimeTable 关系，将函数依赖 (teacherNo, courseNo) → roomNo 分离之后将得到两个关系 CourseTable 和 TeacherRoom，其中，关系 CourseTable 的主关键字为 (teacherNo, prelectionDate, prelectionTime)，关系 TeacherRoom 的主关键字为 (teacherNo, courseNo)。如图 3.44(a) 和 (b) 所示。

teacherNo	courseNo	prelectionDate	prelectionTime
T001	C001	2-Mar-2009	8:30
T001	C001	9-Mar-2009	14:30
T002	C001	2-Mar-2009	8:30
T002	C001	9-Mar-2009	14:30
T002	C004	4-Mar-2009	8:30
T003	C005	5-Mar-2009	8:30

图 3.44 (a) CourseTable 关系

teacherNo	courseNo	roomNo
T001	C001	教西 C106
T002	C001	教西 C206
T002	C004	教西 C106
T003	C005	教西 C108

图 3.44 (b) TeacherRoom 关系

一个关系模式满足 3NF 但不满足 BCNF 的情况很少发生，一般如下情形是一个关系模式满足 3NF，但是可能不满足 BCNF 的常见情况：

关系模式中包含两个或者多个复合候选码；

候选码重叠，也就是候选码具有重叠属性。

3.4.8 关系模型规范化过程

规范化是一种形式化的技术，它利用主关键字或候选关键字，以及属性间的函数依赖来分析关系。这种技术包括一系列作用于单个关系的测试，一旦发现某个关系不满足特定范式的要求，就对该关系进行分解，直至满足特定规范化的要求。通过这种技术可以根据数据库需要的程度对其中的关系模式进行规范化，从而获得良好的性质。

以上从 1NF 到 BCNF 范式的讨论都是基于函数依赖进行的，如果一个关系模式经过改造之后得到的所有关系模式都属于 BCNF，那么在函数依赖的范围内，它已经实现了彻底的分离，并消除了插入和删除的异常问题。因此，一般情况下，工程设计中规范化进行到 BCNF 之后，关系模式中可能存在的大部分问题都将得到解决。当然，满足 BCNF 要求的模式仍然可能存在数据冗余问题，这将涉及到多值依赖的概念，多值依赖也是数据依赖的一种。通过消除多值依赖，可以将关系模式进一步规范化为第四范式。通过消除满足第四范式的模式中的连接依赖，又可以进一步将模式规范化为第五范式。对于第四范式和第五范式，在实际数据库设计中很少能够用到，在此不再讨论。

规范化的执行应该分成一系列的步骤，每一步规范化的结果都对应某一特定范式，随着规范化的进行，关系模式的形式将逐渐变得更加规范，表现为具有更少的数据冗余以及数据更新异常。图 3.45 说明了个范式之间的关系，从图中也可以看出，一个满足高级范式的关系模式必然满足低一级的范式。

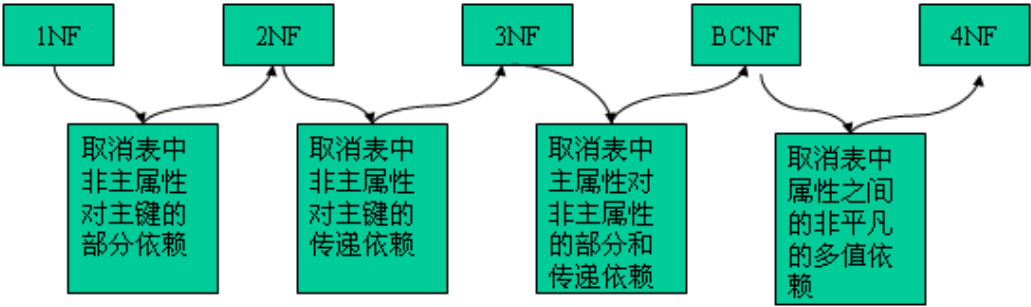


图 3.45 各范式之间的关系

根据以上论述的规范化内容，我们可以将任何不符合 BCNF 的关系分解满足 BCNF 的关系，但是，在所有情况下都将一个关系模式规范化为 BCNF 并不一定是最佳选择。因为在一个属于 3NF 的关系模式向 BCNF 规范化的过程中，可能会丢失一些函数依赖，因而导致一些重要的约束也可能随之丢失，但如果把一个关系模式只规范化为 3NF，则所有的函数依赖都能够保存下来。因此，在数据库设计中应该根据实际情况和需要对关系模式进行规范化到 3NF 或 BCNF。

在以上所讨论的规范化内容中，其目标或者结果是得到一个结构上一致并且拥有最少冗余的逻辑数据库设计。然而，在追求数据冗余最少的时候，却有可能降低数据的访问效率。某些情况下，为了获得某些性能和效率上的提高，有必要放弃部分规范化的要求。

因此，在数据库的物理设计阶段或者监控和调优阶段，有时为了提高系统查询效率，可能会将规范化过程中通过分解得到的关系重新加以合并，人为引入一定数据冗余，这种人为违反规范化从而达到提高系统整体效率的方法称为反规范化的细想。

3.5 数据库设计辅助工具



内容提要

本节通过以上 4 小节内容，对数据库设计整个过程进行了介绍，重点介绍数据库设计过程中的概念设计和逻辑设计阶段，其内容涵盖了数据库设计中的技术和设计过程，它们分别是数据库设计方法学中的两个方面，方法学三个方面的最后一个是工具的使用，或者叫计算机辅助软件工程（Computer Aided Software Engineering, CASE），本节将针对数据库设计辅助工具进行简单介绍。本节主要包含如下内容：

- 数据库设计辅助工具简介；
- Teaching 案例在 PowerDesigner 中的设计过程。

3.5.1 数据库设计辅助工具简介

计算机辅助软件工程（Computer Aided Software Engineering, CASE）已经广泛应用于软件生命周期的各个阶段。从广泛的角度看，CASE 工具是能够支持软件工程的任何工具，它们能够提供自动或者半自动化的软件支撑环境，帮助软件设计人员或者开发人员高效地完成任务。当前支持软件生命周期各阶段的工具和集成开发环境很多，应用也非常广泛，例如，当前面向对象软件设计中经常使用支持 UML 建模的工具，比较知名的有 IBM 公司的 Rational Rose，开放源码的 Eclipse 集成开发环境下面的 UML 工具 EclipseUML 和 UML2 等。

在实际工程应用中，数据库设计人员和数据库管理员同样需要合适的工具，以支持高效的数据库设计与开发。目前支持数据库设计 CASE 工具也很多，例如，PowerDesigner、ERwin、BDB 和 CASE Studio 等，它们目前应用较为广泛，除此之外，由于数据库概念设计技术 ER 模型可以采用 UML 方式进行表示，因此支持 UML 建模的工具一般都能够支持数据库设计，例如前面提到的支持 UML 建模的 Rational Rose 同样支持数据库设计。这些 CASE 工具功能各不相同，但是对于数据库设计来讲，功能和过程大同小异，本书将以 PowerDesigner 为例，介绍 CASE 工具在数据库设计中的应用。

PowerDesigner 是目前最为流行的软件分析工具之一，它支持多种建模技术的设计：数据库建模（概念数据模型 CDM、物理数据模型 PDM），业务处理模型 BPM，面向对象模型 OOM 以及自由模型 FEM。PowerDesigner 在数据建模方面，它利用基于可靠方法，支持真正的两级（概念上和物理上）关系数据库建模，并能够根据设计自动生成数据库模式。

3.5.2 Teaching 案例在 PowerDesigner 中的设计过程

在本小节，我们仅仅通过 Teaching 案例的数据库模型建立过程，简要介绍使用 PowerDesigner 创建和操作概念数据模型 CDM 和物理数据模型 PDM 的方法和过程，从而体验工程中常用的数据库辅助工具的特点。关于 PowerDesigner 工具的具体使用，请读者参考 PowerDesigner 使用手册或者相关资料自学。以下设计过程采用 PowerDesigner12 版本创建概念数据模型 CDM。

1) 运行 PowerDesigner 软件，在主菜单中选择“File”→“New”命令，弹出新建模型窗口，如图 3.46 所示。

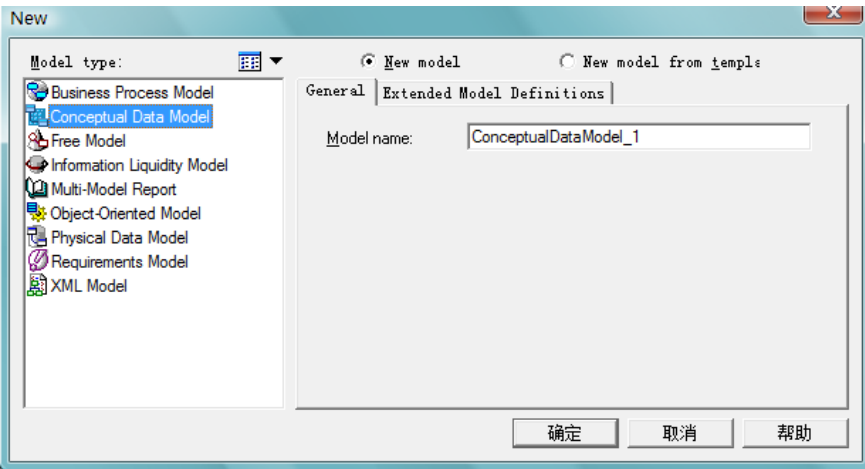


图 3.46 新建概念数据模型

2) 选择 Conceptual Data Model, 并输入模型名称 “Teaching” 单击 “OK” 按钮。在 Workspace 中出现一个概念数据模型, 如图 3.47 所示。

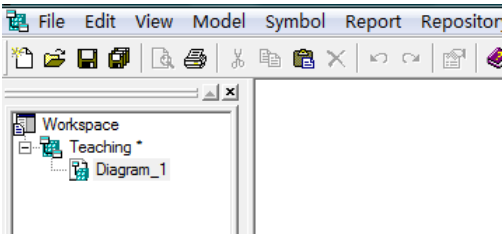


图 3.47 编辑新建模型属性

3) 鼠标右键单击 Diagram_1, 选中 Properties 选项, 弹出属性配置菜单, 输入将要建立的学生视图的名称 Student, 如图 3.48 所示。

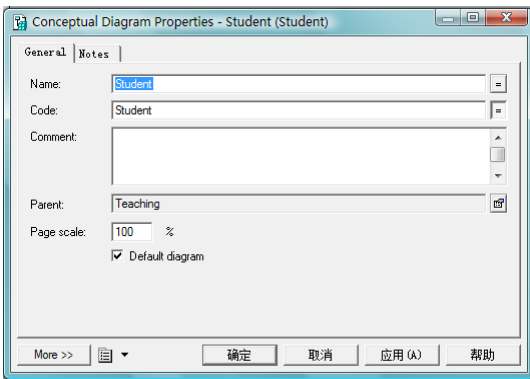



图 3.48 编辑视图属性

4) 在概念模型中创建实体。单击 Palette 面板中的 “Entity” 图标  , 在模型区域单击鼠标左键, 在鼠标单击的位置出现 Entity 的图符, 默认的命名方式为: Entity_n, 其中 n 为当前实体在创建中的次序。如图 3.49 所示。其中, 第一栏 Entity_n 为实体名, 实体的属性建立后将放在第二栏, 实体中若定义了标识符, 将在第三栏显示。

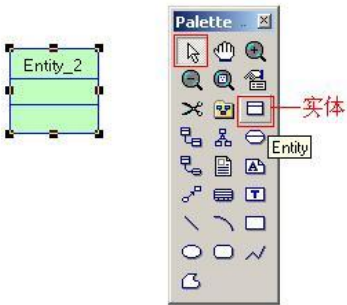


图 3.49 创建实体

5) 鼠标左键双击实体图标, 弹出实体的属性窗口, 在此编辑实体属性。如图 3.50 和图 3.51 所示。

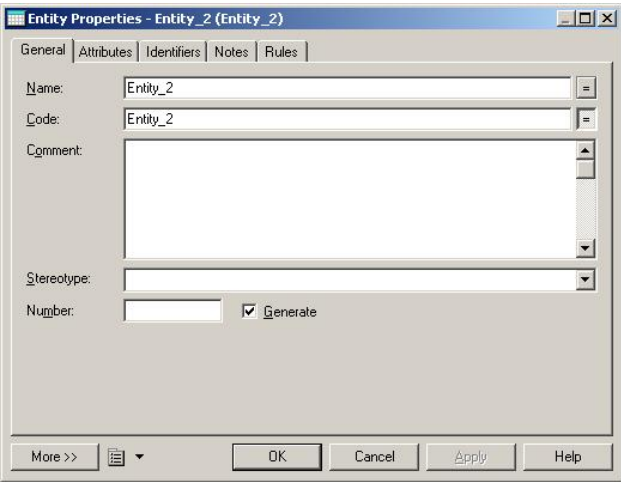


图 3.50 编辑实体基本属性

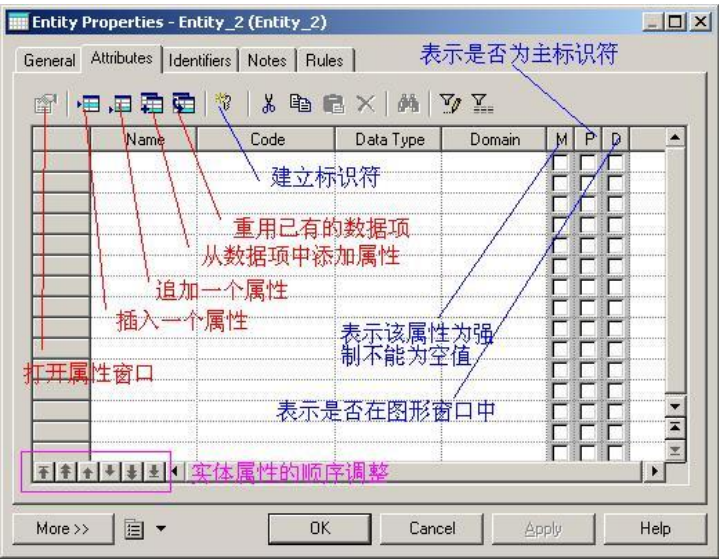


图 3.51 编辑实体属性

6) 对于 Teaching 案例中的学生视图 Student，创建实体图 3.52 所示。

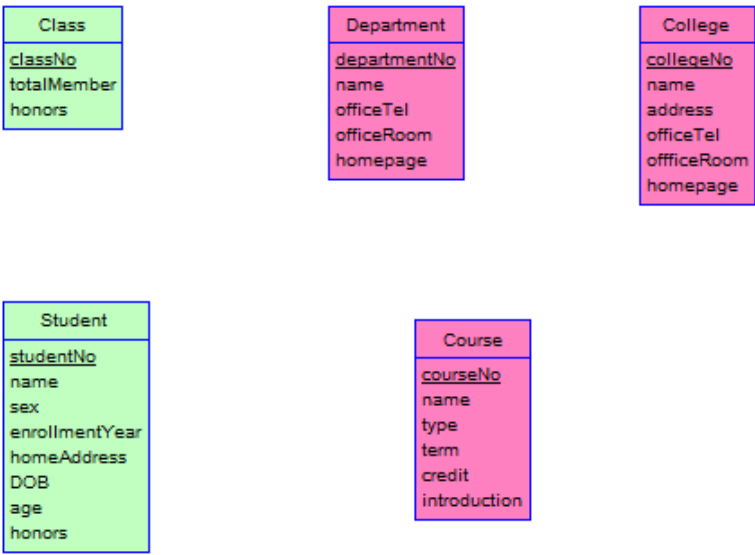


图 3.52 Teaching 案例 Studentn 视图实体

7) 在概念模型中创建实体间联系。单击“实体间建立联系”工具，单击一个实体，在按下鼠标左键的同时把光标拖至别一个实体上并释放鼠标左键，这样就在两个实体间创建了联系，右键单击图形窗口，释放 Relationship 工具，如图 3.53 所示。



图 3.53 创建联系

8) 鼠标左键双击联系图标，弹出联系属性窗口，在该属性窗口中可以编辑联系属性，如图 3.54 所示。在联系属性窗口的 Cardinalities 选项中，可以设置联系的基数约束与参与性约束，如图 3.55 所示。

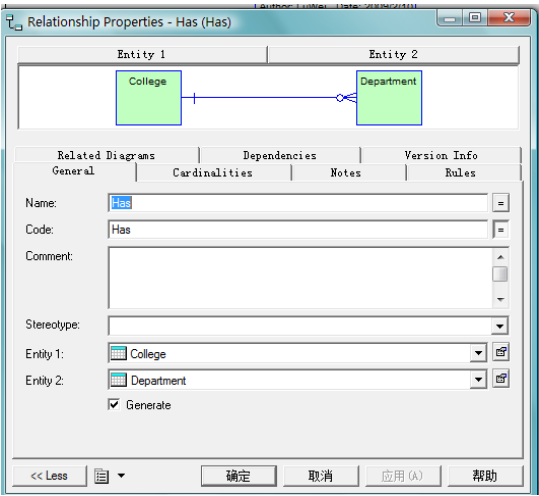


图 3.54 编辑联系基本属性

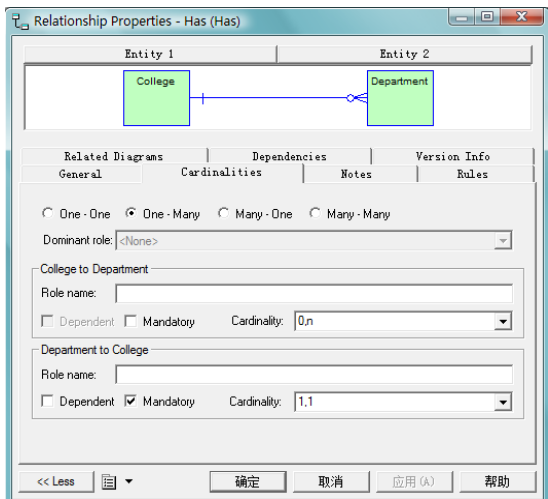


图 3.55 设置联系约束

9) 本书 Teaching 案例的学生视图 Student 在完成实体和联系之后, 结果如图 3.56 所示。由于 PowerDesigner 中不支持联系的属性, 在此, 使用一个关联 Association 代替联系属性, 并用虚线与对应联系连接。

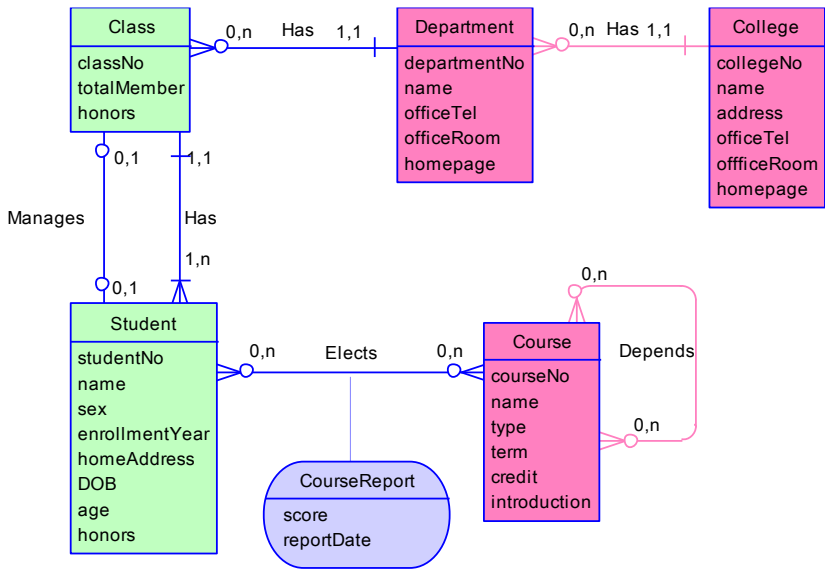


图 3.56 Teaching 案例 Student 视图的 ER 模型

10) 按照同样过程, 可以建立本书 Teaching 案例中教师视图 Teacher 的 ER 模型图, 如图 3.57 所示。

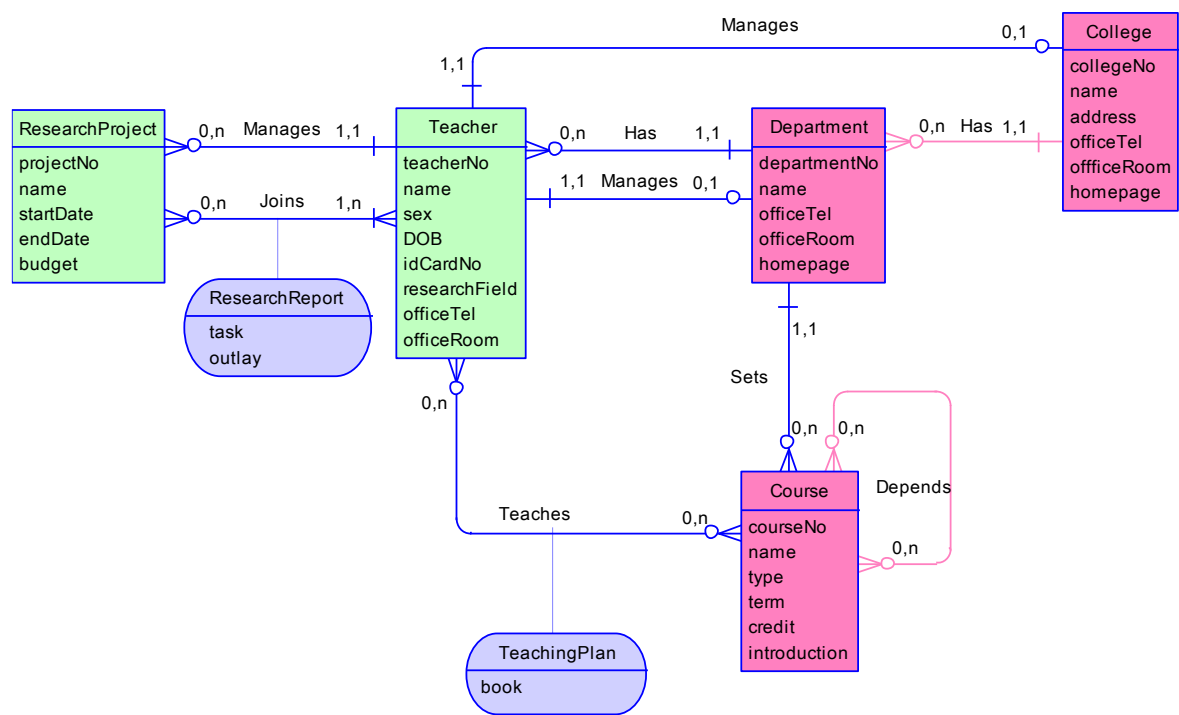


图 3.57 Teaching 案例 Teacher 视图的 ER 模型

11) 按照逻辑数据库设计内容中的步骤，将以上两个视图去除其中与关系模型不兼容的特征，包括多对多联系、多对多递归联系、多值属性以及多元联系之后，得到的 ER 模型图如图 3.58 和 3.59 所示。

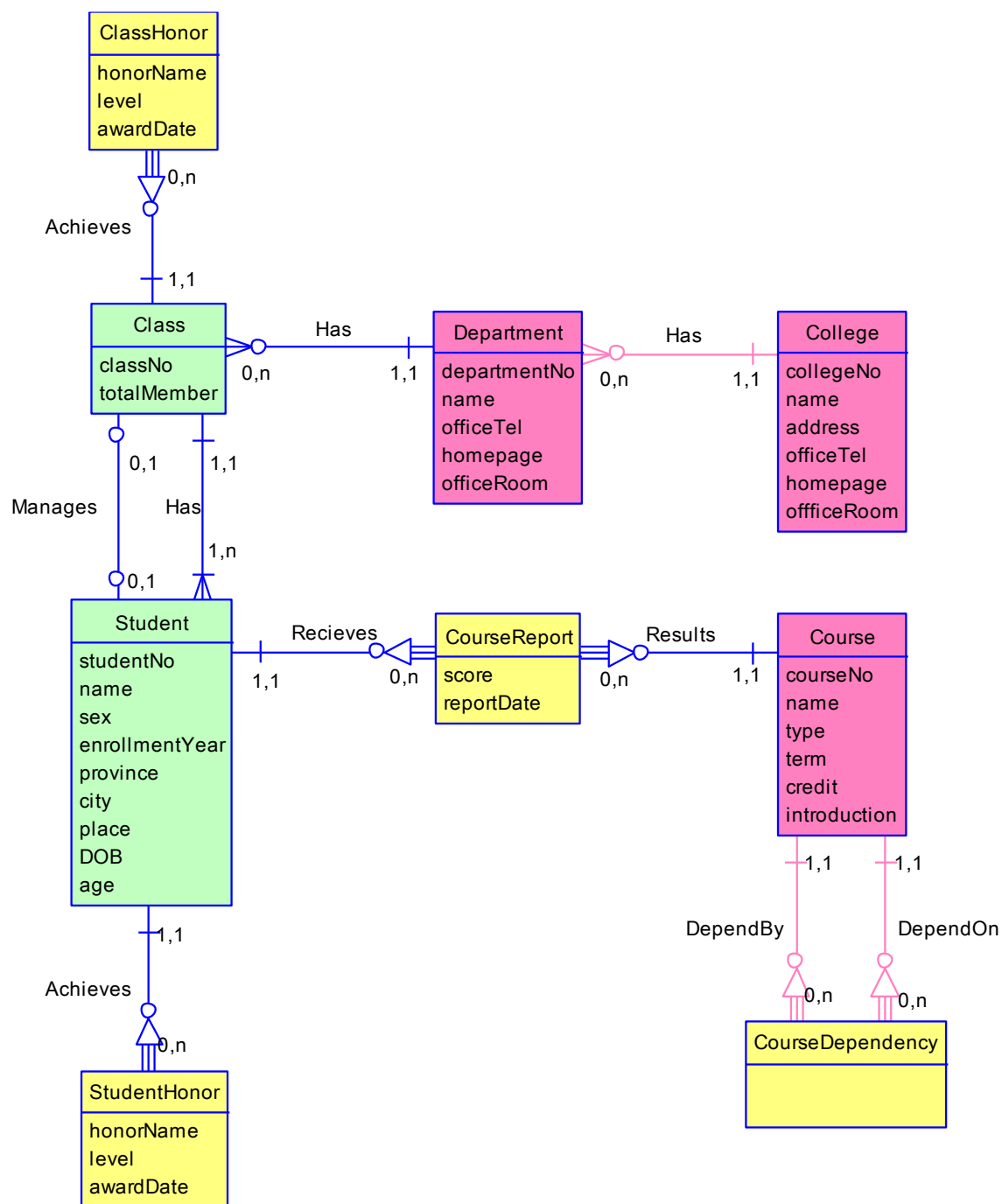


图 3.58 Teaching 案例 Student 视图最终 ER 模型

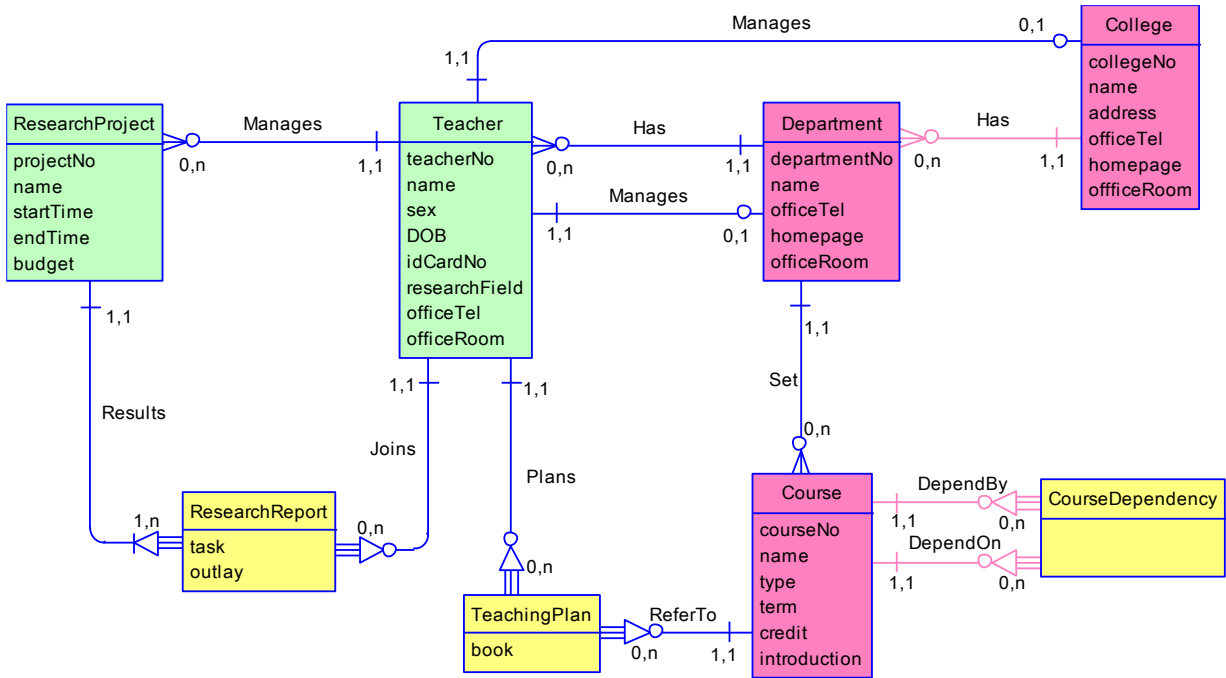


图 3.59 Teaching 案例 Teacher 视图最终 ER 模型

12) 将 Student 视图与 Teacher 视图合并之后得到的全局 ER 模型图如图 3.60 所示。

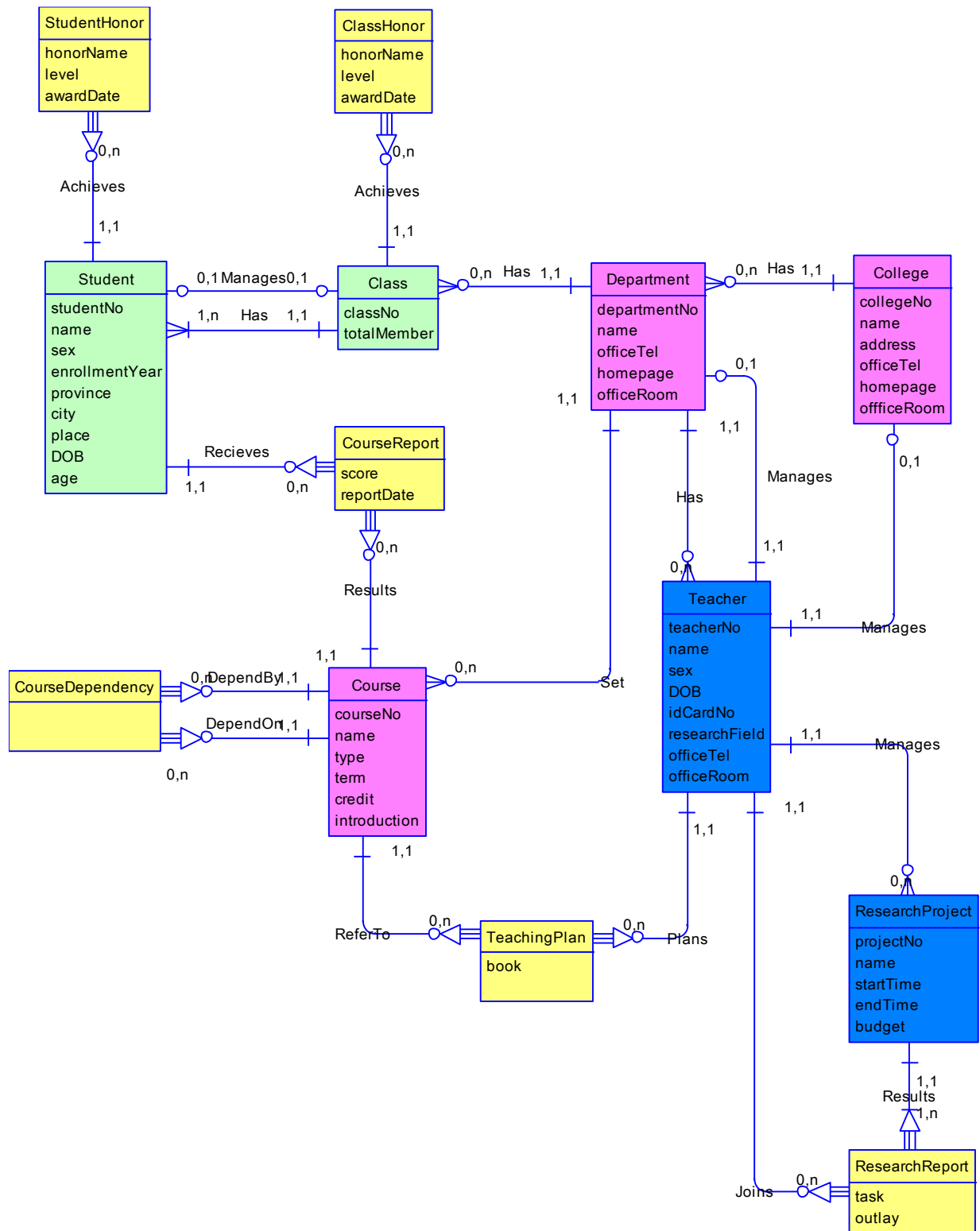


图 3.60 Teaching 案例的全局 ER 模型

13) 将全局 ER 模型图映射为关系，得到的逻辑模型如图 3.61 所示。

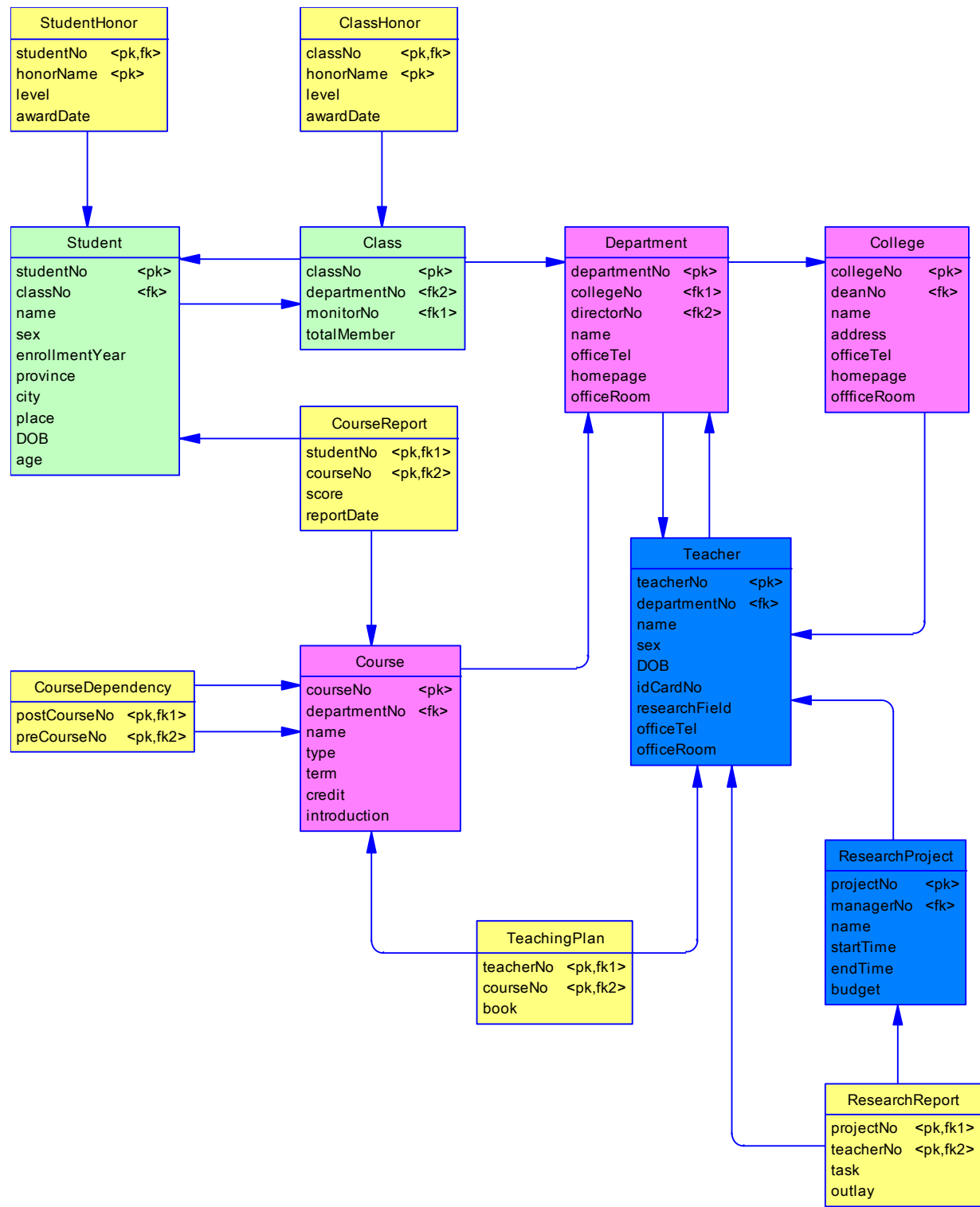


图 3.61 全局关系模型图



本章小结

数据库设计这一章主要介绍了数据库系统的生命周期，并针对其中数据库设计阶段展开讨论。在数据库设计中，重点针对概念设计和逻辑设计两个主要阶段，从技术和过程两个方面详细论述了数据库设计方法以及注意事项。

通过本章学习，应该掌握数据库概念设计主要技术—ER 模型技术，并能够根据用户需求，按照工程化的方法和步骤，建立概念数据库模型。掌握概念模型向逻辑模型映射的方法，并能够根据概念模型，按照工程化的方法和步骤，将其映射为关系模型。掌握关系模式规范化的思想，能够对给定关系模式作出评价。

本章在论述过程中采用了大量的案例，这些案例对理解所论述的内容具有很好的帮助，代表了数据库设计中的常见情况。



本章练习

