第八章 数据库设计

8.1 数据库设计概述

要将现实世界中客观存在的事物以数据的形式存储到计算机中并进行处理，就需要对其进行分析、抽象，进而确定数据的结构以及数据间的内在联系，这一过程称为数据库设计。在中小型规模的实际开发中，数据库设计并不困难，但是却非常重要。如果数据库设计的不好，很容易出现数据冗余、效率低下以及数据库系统不可靠等问题。而且，客户端应用程序是依赖于数据库设计的，一旦数据库设计上存在问题将来补救起来通常会很困难。

数据建模

具体来说，在设计数据库时，设计者必须首先规划好要在数据库中存储关于哪些事物的信息，以及要保存关于其中各种事物哪些方面的信息，此外，还需要确定这些事物内部或其之间的相互关系。比如，在学校的教学管理系统中，需要保存关于学生和课程的信息；其中，关于学生需要记录其学号、姓名、性别、出生日期共四方面信息，关于课程要记录其课程编号、课程名称、课时数共三方面的信息；此外，学生的学号不允许出现重复，课程名称不能为空，学生可以选修课程，且多个学生可以选修同一门课程、而一个学生也可以同时学习多门课程、并且要记录其考试成绩相关相关信息（学号、课程编号和考试成绩）。上述过程也称为“数据建模”（Data Modeling），即根据应用开发的需要建立数据模型的过程。

在数据库设计中，设计者建立的数据模型应满足三个方面要求：

- 数据模型应能够比较真实地模拟现实世界
- 数据模型应容易为人所理解
- 数据模型应便于在计算机中实现

数据模型三要素

在进一步讲解之前，我们先分析一下数据模型的三个组成要素：

- 数据结构

数据结构描述的是客观事物的静态特性，比如关于学生信息包括学号、姓名、性别和出生日期四个属性组成，当然具体记录哪些事物的哪些方面的属性要根据实际应用的需要，这很类似于面向对象编程时类中的属性定义。

需要强调的是，这里的所谓“客观事物”可以是具体的“物”（如人、兽、鸟），也可以是抽象的“事”（事件、现象或行为，如潮起潮落、云卷云舒）。学生选修某一门课程，严格说这是一种事件，但根据应用的需要，我们也可以记录其相关信息，比如哪个人选了哪门课、以及其考试成绩等。

- 数据操作

数据操作描述的是事物的动态特性，比如人的姓名是可以修改的、考试成绩可以被录入、更

改和删除。这些操作类似于 Java 面向对象编程语言中，类里面定义的成员方法，比如最常见的提供属性值存取操作的 setter/getter 方法。不同的是，在数据库中对数据的操作并不体现在数据表结构中，而是以在 DBMS 层面运行 DML 指令的方式来实现。

- 完整性约束

完整性约束描述的是事物内部和事物间的约束性关系，引入完整性约束是为了确保数据模型能够满足实际应用的需要。比如应用业务逻辑要求学生的学号必须是唯一的、图书的定价不不允许为负值、工资表中出现的员工编号取值必须在员工信息表中出现过等等。完整性约束主要分为域完整性约束、实体完整性约束和参照完整性约束三种，这些我们前面已接触过（参见第 7.2.6 节），也可以根据需要由用户自行定义。

三个世界

数据建模过程实际上就是根据应用业务逻辑的需要，将现实世界中的信息进行分析、抽象、提取并转换为与计算机和 DBMS 相关的数据模型的过程。在整个数据建模过程中，信息的状态（也可以说是数据模型的状态）可抽象归纳为现实世界、概念世界和机器世界三个层次，如图 8-1 所示。

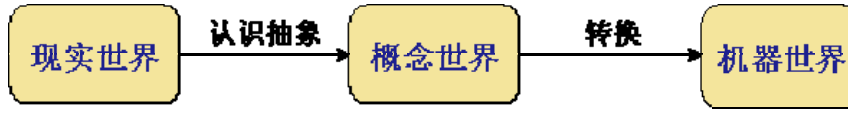

图 8-1 数据模型层次

现实世界即我们生活于其中的世界，现实世界中事物以及事物间的联系是客观存在的。

概念世界是人们对现实世界中客观事物及其联系的认识和抽象。　　概念世界中按用户的观点对现实世界建模，所得到的“概念数据模型”（详见第 8.2 节）不依赖具体的计算机系统和 DBMS。

在机器世界中，按计算机的观点对概念世界中的事物（实体及实体间的联系）进一步建模，将之抽象/转换为与计算机/DBMS 相关的“物理数据模型”（详见第 8.4 节）。

表 8-1 给出了三个世界中的数据模型常用术语及其对应关系，后文中会做详细介绍。

表 8-1 数据模型术语及对应关系

现实世界	信息世界	复杂视图
客观事物	实体	记录
客观事物及其联系	实体模型	数据模型
特性	属性	字段
特性定量描述	属性值	字段值
特性描述的范围	域	字段取值范围
关于客观事物特性的描述集合	实体类型	记录类型
某一类客观事物的集合	实体集	数据表
唯一标识客观事物的特性	标识属性	候选关键字（唯一键）

非唯一标识客观事物的特性	非标识属性	次关键字
选定的唯一标识客观事物的特性	选定的标识属性	主关键字（主键）

8.2 概念数据模型

前文中已提及，在设计数据库时，设计者应该首先规划好要在数据库中存储关于哪些事物的信息，以及要保存关于其中各种事物哪些方面的信息，此外，还需要确定这些事物内部或其之间的相互关系。这种在概念世界中、按照用户的观点对现实世界中客观事物及其联系的认识和抽象，所得到的数据模型被称为“概念数据模型”（Conceptual Database Model，CDM）

概念数据模型（为简洁起见，以下将简称为 CDM）是从用户的视角出发对信息进行抽象而建立的模型，它并不依赖于具体的计算机系统或 DBMS 系统，CDM 主要用于数据库的概念设计。

CDM 以实体-关系模型（Entity-Relationship Model，E-R Model，详见第 8.2.3 节）为基础，将现实世界中的客观对象抽象为实体和关系。到机器世界中，CDM 将被转换为特定 DBMS 所支持的物理数据模型（Physical Database Model，PDM，详见第 8.3 节）。

CDM 相关术语主要包括实体、属性和关系等，下面分别加以介绍。

8.2.1 实体

实体（Entity）是客观存在并且可以相互区分开来的事物，比如张三、李四是两个不同的实体，一件商品、一个部门、一次购物行为都是一个实体。同一类实体组成的集合则称为实体集（Entity Set），比如一个单位的全部员工（每个员工是一个实体）就组成了员工实体的集合。在数据库中，每个实体对应的是一条记录行，而实体集则对应于数据表。

每个实体都会包含若干个属性（Attribute），以描述实体的相关特性，比如在员工实体中，一般会包含其工号、姓名、职位以及其它与特定员工相关的特性信息。在具体的应用开发项目中，要在数据库中存储哪些实体、以及存储实体的哪些属性信息，要由具体的应用业务逻辑决定。为直观起见，在概念数据模型中，可以使用图形化方式表示实体，如图 8-2 所示。

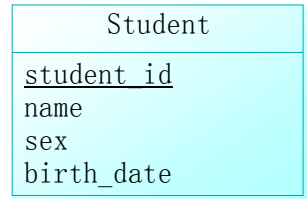

图 8-2 CDM 实体模型

可以看出，图中以矩形框表示实体，并在矩形框内部列出该实体的相关属性。其中，属性 student_id 名称下的下划线用于标明该属性为标识符（Identifier），所谓标识符是指那些可用于唯一地标识一个实体的属性或属性组合，比如在上述 Student 实体中，student_id 属性可以唯一地标识一名学生，而所有其它属性都只保存与该学生相关的信息。换句话说，根据学号可以唯一地确

定一名学生的姓名、性别和出生日期等信息，而根据其它属性则不能，比如可能存在两个或多个学生同名的情况，但可以确保其学号是各不相同的。

做为一种良好的习惯，在构建概念数据模型时，我们应为每种实体都创建一个标识符，这些标识符在将来的数据库表中将成为主键（主键相关知识参见本书第 7.2.4 节），以用于唯一标识表中的每一行记录，并在数据库访问时提高操作性能。

8.2.2 关系

在概念数据模型中，所谓关系（Relationship）指的是实体之间的对应关系，实际上其描述的是现实世界事物之间的相互关联。比如学生和课程实体之间存在着学习关系、部门和员工之间存在着隶属关系、顾客和商品之间存在着购买关系。在数据库中，实体间关系一般会表现为外键约束（外键相关知识参见本书第 7.2.5 节），但也可能被转换为独立的数据表，比如学生和课程之间的学习关系可以被单独定义为“学生选课信息表”，本节中会对此做详细讲解。

关系的基数

按照关系中所涉及的实体的数量，可以将实体见关系分为如下三种：

- 一对一关系

两个实体集 A 和 B，若 A 中的每个实体至多和 B 中的一个实体有联系，反之，B 中的每个实体至多和 A 中的一个实体有联系，称 A 对 B 或 B 对 A 是一对一关系。比如，每个部门至多只能由一个员工（部门经理）负责管理、而每个员工至多只能担任一个部门的经理职务。具体如图 8-3 所示。

图 8-3 一对一关系模型

其中，方框之间的连线用于表示这两个实体集之间存在关系，连线上的靠近实体集方框的小圆圈用于标明该实体集内的实体在另一个实体集中不必须存在对应的实体。即有的部门可能没有部门经理，而有的员工可能不担任任何部门的经理职务。

- 一对多关系

两个实体集 A 和 B，如果 A 中的每个实体可以和 B 中的 0~多个实体有联系，而 B 中的每个实体至多和 A 中的一个实体有联系，则称 A 对 B 为一对多关系。比如，每个部门可以包含多名员工（也可能包含 0 个员工——部门刚组建），但每个员工至多只能隶属于一个部门（也可能不隶属于任何部门——尚未分派工作），具体如图 8-4 所示。

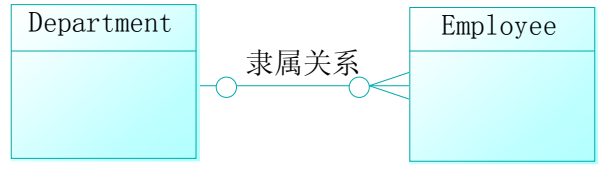

图 8-4 一对多关系模型

说明：图中右侧靠近 Employee 实体方框的鸟爪状图形用于标明 Department 实体集中的一个实体可以与 Employee 实体集中的多个实体相关联，相应地，左侧靠近 Department 实体方框的小圆圈则标明 Employee 实体集中的一个实体可以与 Department 实体集中的 0~1 个实体相关联。

- 多对多关系

两个实体集 A 和 B，若 A 中的每个实体可与和 B 中的多个实体有联系，反之亦然，称 A 对 B 或 B 对 A 是多对多关系。比如学生和课程之间的关系，每个学生可以选修 0 ～多门课程，而每门课程可以被 0 ～多名学生选学。此时，两个实体集方框间的关系连接线上需要加上两个鸟爪状图形来标记，具体如图 8-5 所示。

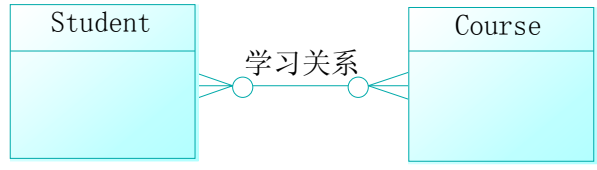

图 8-5 多对多关系模型

强制关系

如果一个实体集 A 中的每个实体在另一个实体集 B 中至少存在一个对应的实体，则应在关系连线上靠近 B 的位置使用一段短的交叉线、而不是先前的小圆圈来标记标记。比如，每个部门必须也只能由一个员工（部门经理）负责管理、而每个员工至多只能担任一个部门的经理职务（也可能不管理任何部门）的情况，则应将图 8-3 改为如图 8-6 的形式。

图 8-6 强制性一对一关系模型

此时，我们称从 Department 实体集到 Employee 实体集的关系是强制的。相应地，由于一个员工可能不管理任何部门，因此从 Employee 实体集到 Department 实体集的关系为可选的。

类似地，如果每个学生必须选修 1～多门课程，而每门课程可以被 0 ～多名学生选学，则应将图 8-5 改为如图 8-7 的形式。

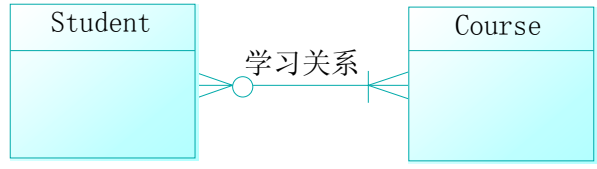

图 8-7 强制性多对多关系模型

强制关系也可以是双向的，比如每个学生必须选修 1 门以上（1～多门）的课程，而每门课程都有 1 名以上（1～多名）学生选学，如图 8-8 所示。

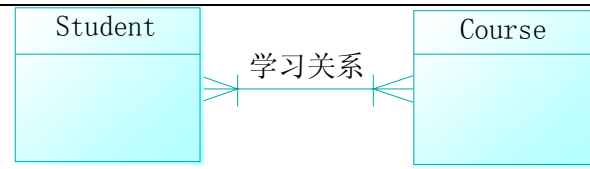

图 8-8 双向强制性多对多关系模型

为便于记忆，表 8-2 归纳了各种可能出现的关系基数类型及其图标。

表 8-2 关系基数标识

图 标	关系基数	说明
	(0 , 1)	另一实体集中的每个实体在本实体集中可以有 0~1 个实体相对应
	(1 , 1)	另一实体集中的每个实体在本实体集中必须也只能有 1 个实体相对应
	(0 , n)	另一实体集中的每个实体在本实体集中可以有 0~多个实体相对应
	(1 , n)	另一实体集中的每个实体在本实体集中必须有 1~多个实体相对应

注：表中所谓的“本实体集”是指距离图标较近的实体集方框所表示的那个实体集。

反身关系

有些情况下，同一个实体集内部的实体之间也可能存在着关联关系，比如员工与其上司（其它的员工）之间存在着管理和被管理的关系，这种关系被称作反身关系，关系的两端都连接到同一个实体集，如图 8-9 所示。

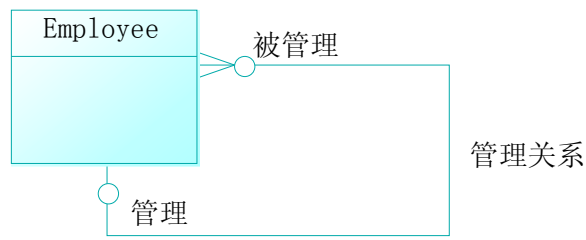

图 8-9 强制性多对多关系模型

上图具有如下两方面的含义：

- 一名员工最多只有一个（直接）上司（也可能没有，比如单位的总经理没有上司）
- 一名员工可以管理 0～多名其它的员工

Oracle 数据库 Scott 方案下的样本数据表 Employee 描述的就是这种情况。

将多对多关系转换为实体

在多对多关系中，可能会出现有属性与关系相关联、而不是单纯的与实体相关联的情况，将这样的属性添加到任何一个实体中都是不合理的，此时应将该关系转换为（或者说定义为）实体。比如，学生与课程之间的多对多关系（参见图 8-5），加入我们还想记录学生的学习成绩，该属性就只能与两种实体间的学习关系相关联——每个成绩都依赖于其所涉及的学生和课程。此时，可

以将学习关系定义为实体，以更有效地表示当前的状况，如图 8-10 所示。

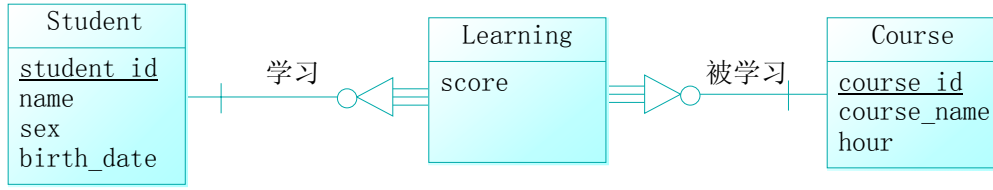

图 8-10 多对多关系转换为实体

从图 8-10 中可以看出，这实际上是将一个多对多关系转换成了两个一对多关系。Learning 实体集中的每个实体都必须在 Student、Course 实体集中各存在一个对应的实体（两个强制关系），因为学习关系只有在与一个特定的学生和课程实体相关联时才有意义。换句话说，新建的 Learning 实体既依赖于 Student 实体、同时又依赖于 Course 实体，后两者中的标识符组合起来则可以唯一地标识 Learning 实体集中的一个实体（一个学号 student_id 和一个科目编号 course_id 合在一起可以唯一地确定一个考试成绩 score），Learning 实体集方框两侧小圆圈和鸟爪状标记中间的三角形就是用来标明这种依赖性的。

说明：由于已使用三角形标记标明了 Learning 实体集中的每个实体都依赖于一个特定的 Student 实体和 Course 实体（这样可以更清晰地表示实体集之间的依赖性），因此不需要在 Learning 实体集方框中再添加 student_id 和 course_id 属性。

将多对多关系转换为实体后更适合其相关属性信息在数据库中的存储，这样可以很好地避免数据冗余，在实际开发中提倡进行这样的处理。比如上面的 Learning 实体集对应的就是“学生选课信息表”（或“学生考试成绩表”），其中以外键的形式引用了“学生个人信息表”（与 Student 实体集相对应）和“课程信息表”（与 Course 实体集相对应）的主键字段（student_id 和 course_id），这两个外键字段同时又构成了 Learning 表的联合主键，以用于唯一地标识表中的每一行记录。

关系的完整性

关系的完整性（Relationship ReIntegrity）指的是实体或实体间关系必须满足的某种约束条件或规则，通常包括实体完整性、参照完整性和域完整性三种，引入关系完整性是为保证数据库中数据的正确性和相容性，具体表现为数据库中各种类型的约束（参见本书第 7.2.6 节）。这里介绍的是关系完整性在概念数据模型层面的表现。

- 实体完整性

实体完整性（Entity Integrity）是指实体集中的每一个实体都应该能被唯一地标识。这是因为实体集中的实体（对应数据库表中的记录）对应现实世界中的某一特定事物，而现实世界中的事物是可以区分的、而且都用有其各自的唯一性标识。例如，每个人都有其唯一的身份编号、员工有其唯一的员工编号等。**实体完整性的具体表现是，实体集中应存在做为每一个实体的唯一性标识的标识符（对应数据库表中主键），标识符的取值必须唯一、且不能为空值**。否则的话，实体集中就可能存在不可标识的实体，比如学生信息表中，学号属性（标识符）不能为空值也不应出现重复值，否则无法对应到某一个特定的学生，保存这样的信息没有意义。

在数据库中，实体完整性表现为主键和唯一键约束。

- 参照完整性

参照完整性（Referential Integrity）是指两个实体集（也可能是一个实体集）中的对应属性的

属性值应应保存一致。比如，工资信息表中的员工编号取值必须是在员工信息表中出现过的（对应在数据库中，子表中的外键字段取值参照主表中的主键或唯一键字段）。引入参照完整性就是为了确保表间的数据一致性，以防止因操作疏忽而导致的数据丢失或出现无意义的数据。

在数据库中，参照完整性表现为外键约束。

- 域完整性

域完整性（Field Integrity）也称用户定义完整性，是指实体属性值域的完整性，包括字段的数据类型、格式、取值范围、是否允许空值等。域完整性约束将目标属性的值限制在指定的范围内，例如，员工编号不允许为空值 NULL，人的年龄必须界于 0～120 岁之间，人的姓名不能少于 4 个字符等。

在数据库中，域完整性表现为非空和检查约束。

8.2.3 E–R 模型与 E–R 图

实体-关系模型（Entity-Relationship Model, E-R Model），简称 E-R 模型（也称 E-R 方法），由美籍华人 P.P.S.Chen 于 1976 年提出，是概念数据模型的典型代表。

在 E-R 模型中，现实世界是由一切称为实体的对象和这些对象之间的关系组成的，其三个组成要素就是前述的实体、关系和属性。E-R 模型中使用 E-R 图（Entity-Relationship Diagram）来描述现实世界的概念模型，在 E-R 图中：

- 使用矩形框表示实体
- 用椭圆形表示属性，并用连线将属性与其所属的实体连在一起
- 用菱形框表示关系，并用连线分别与相关实体相连，且需在连线上注明联系类型。

具体如图 8-11 所示。

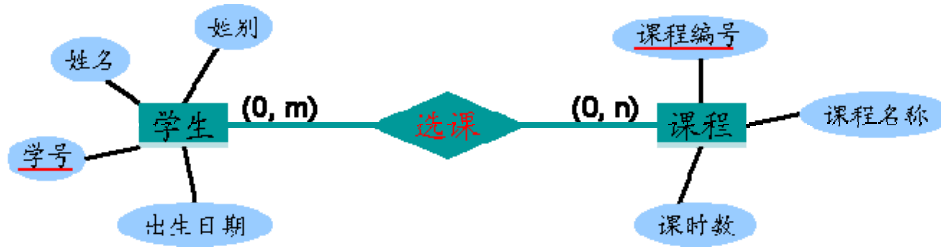

图 8-11 多对多关系 E-R 图

8.3 物理数据模型

物理数据模型（Physical Data Model，PDM）描述的是数据在具体数据库中的组织/存储结构，因此 PDM 和具体的 DBMS 相关，同时还和操作系统类型硬件结构有关，否则无法真正实现数据在数据库中的存储。

做为对比，概念数据模型（CDM）描述的则是现实世界的概念化结构，即从用户的视角出发对信息进行抽象而建立的模型，它并不依赖于具体的计算机系统或 DBMS 系统。引入 CDM 的目的是使设计人员在数据库设计的初始阶段，抛开 DBMS、计算机操作系统以硬件结构的差异等具

体技术问题，集中精力分析数据以及数据之间的联系等。但最终 CDM 都要转换为相应的 PDM，以真正实现数据在具体数据库中的存储。

简单地说，PDM 建模的主要任务是确定所有的表结构、字段类型和约束条件。比如在表中定义主键以唯一标识其中的每一行记录、定义外键用于确定表间（字段之间）的引用关系、以及根据需要定义其它类型的约束规则（如果非空和检查约束等），并根据 tu 需要进行可能的数据库范式化容。说了这么多，下面给出一个具体的物理数据模型范例，如图 8-12 所示。

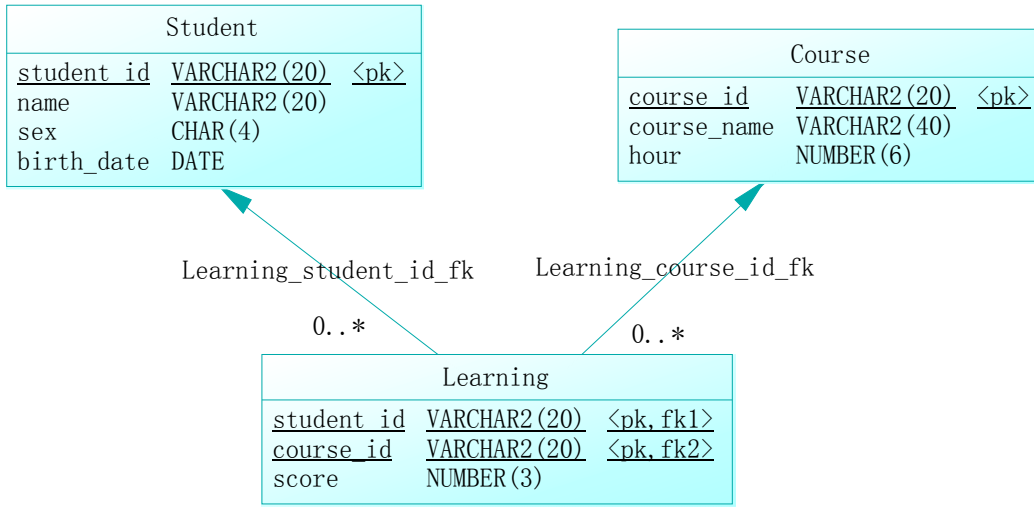

图 8-12 物理概念模型图

说明，图 8-12 所示的 PDM 是基于 Oracle9i2 数据库的，其中矩形方框描述的是数据库中的表，具体包括表名、字段名称、字段类型和约束条件等信息，pk 用于标明主键字段、fk 用于标明外键。箭头标明的则是数据表间存在的引用关系——子表中的外键字段引用主表中的主键/唯一键字段。箭头指向的为主表、箭头起始所在的为子表；箭头中部的文本标识的是引用关系的名称，或者说是子表外键的名称；箭头连线靠近子表位置的 0..*用于标明引用关系的基数——主表的主键/唯一键字段被子表中的 0～多行记录引用，比如，一个学生可以学习 0～多门课程（Student 表中的一个 student_id 字段值可以在 Learning 表中出现 0～次）。在 PDM 中，可供选择的引用关系基数包括 0..*，0..1，1..*和 1..1 四种，结合 CDM 中关系的基数相关讲解，请读者自行体会其对应关系。

简单分析后不难看出，图 8-12 给出的 PDM 与图 8-10 给出的 CDM 相对应，只不过 CDM 中的实体到 PDM 中被转换成了数据表，实体中的属性被转换为数据表中的字段，实体标识符被转换为主键，实体间的对应关系（联系）被转换成表间的外键引用。需要注意的是，PDM 中字段的数据类型均应是明确而具体的，且与具体的数据库相关。比如上述 PDM 中出现的可变长度字符类型 VARCHAR2 就是 Oracle 数据库中特有的。

尽管 PDM 的具体实现与具体的 DBMS、操作系统和硬件相关，学习者可以放心的是，我们在数据库设计时只需给出 PDM 的定义，其底层实现则由具体的 DBMS 自动完成。

8.4 数据库规范化

8.4.1 数据库规范化概述

在数据库设计层面经常会涉及到“规范化”、“范式”等概念，多数应用开发人员认为这些东西过于“理论化”，嫌其繁琐不愿细究、并在日常应用中将之抛到一边。实际上，规范化与否对数据库系统的性能影响显著，而掌握数据库规范化的基本原理，并将之应用到日常数据库设计中并不没有多复杂。

什么是数据库规范化?

所谓数据库规范化（Database Standardization），就是对数据库中所存储的数据进行有效组织的的过程，其主要目的在于避免出现数据冗余和操作异常，进而节省存储空间并更好地确保数据的一致性。

范式

经过多年的积累，数据库业界制定了一系列确保数据库规范化的设计规则，在关系型数据库中，这些规则被称为范式（Normal Form）。数据库范式按照其严格程度从低到高共分为五种❶——第一范式（简称 1NF）、第二范式（2NF）、第三范式（3NF）、第四范式（4NF）和第五范式（5NF）。其中，第一范式需要满足的要求最低，第二范式在第一范式的基础上增加了新的要求，以此类推。其中，第三范式在系统性能、扩展性和数据完整性方面都达到了最好的平衡（最严格的不一定是最可取的，套用老祖宗的话来说就是“过犹不及”），一般关系模式设计要求达到 3NF。因此在实际数据库设计中，最常见的是第一范式、第二范式和第三范式，为便于读者理解和掌握，下面分别对这三个范式做通俗而实用的讲解。

8.4.2 第一范式

对于关系型数据库而言，第一范式是其应遵守的最基本规范化规则，简单说来，第一范式要求实体中的每个属性都必须是不可再分的数据项，或者说，属性必须具有原子性。此外，第一范式还要求为各个相关数据组（实体集）创建独立表格，并以标识符（唯一键或主键）来识别实体集中的每一个实体（数据表中的每一行记录）。

最简单的情况，比如在员工实体中，员工编号、姓名、雇佣日期等相关信息均被分开定义为单个的属性，这很容易理解。

引申一下，员工的联系电话如果存在固定电话、移动电话两种的话，则应分解为两个不同的属性（如 phone、mobile_phone），否则不利于在将来的应用业务逻辑中可能的分化处理需求（比如只获取所有的移动电话号码，并向其发送手机短消息）。需要的话，固定电话还可再细分为住宅电话和办公室电话两项，这也比较容易理解。

再看复杂一点儿的情况，在图书出版或销售领域的数据库中，需要记录图书及其作者之间的对应关系，如果每本书可以有 1～多个作者（为简单起见，假定只需记录作者的姓名），初学者很可能会创建如表 8-3 所示的数据库表。

❶ 此外还包括 BC 范式（Boyce-Codd Normal Form，BCNF），通常认为 BCNF 是修正的第三范式，有时也称为扩充的第三范式。

表 8-3 图书信息表

图书编号	书 名	…	作者 1	作者 2	作者 3	作者 4
b001	…	…	张三	李四	王五	
b002	…	…	赵六			
b003	…	…	麻七	张三		
b004	…	…	王五	赵六	郭八	钱九

为直观起见，这里直接给出了样本表结构及演示数据。可以看出，表 8-3 中的作者 1～作者 4 属于重复字段，他们在逻辑上的性质（或者说含义）与用途完全相同，这不同于住宅电话和办公室电话之间的关系。这样的数据结构定义会导致一系列的问题——如果某本书只有一位作者，那么表中的作者 2～作者 4 三个字段纯属浪费数据库存储空间；此外，在每一次的使用中，还需要附加的逻辑以判断这四个字段所存储的数据值的有效性；更要命的是，如果遇到书的作者人数超过四个时该怎么办，再修改表的结构（添加新的字段）显然不是办法。

此时，数据库设计新手往往会想到的“高明”的解决办法是，将表 8-3 改为如表 8-4 所示的形式。

表 8-4 图书信息表 2

图书编号	书 名	…	作者
b001	…	…	张三，李四，王五
b002	…	…	赵六
b003	…	…	麻七，张三
b004	…	…	王五，赵六，郭八， 钱九

即多个作者姓名保存到一个字段中，中间使用约定的分隔符进行分隔。表面看来，此时不需要修改表的结构、且似乎更灵活地利用了存储空间，但问题是这会导致新的缺陷——如果要增加或删除某一位作者，或者哪怕是基于特定作者姓名进行查询，其操作实现起来都是非常复杂的。出现这种问题的根本原因是，表 8-4 中“作者”字段逻辑上仍是多值的，或者说其字段并非不可分割的数据项，即违背了数据库规范化的第一范式。

规范的解决办法是，将图书与作者之间的这种对应关系定义成独立的实体，即将作者字段从图书信息表中移除、并单独定义图书－作者对应关系表，如表 8-5 所示。

表 8-5 图书－作者信息表

图书编号	作者
b001	张三
b001	李四
b001	王五

b002	赵六
b003	麻七
b003	张三
…	…

此时，如果要为某一本图书添加或删除某一作者、或者查询某一作者编著的所有图书信息等操作就变得非常简单了。第一范式（1NF）是数据规范化的最低要求，在实际数据库设计中是我们必须遵守的。

8.4.3 第二范式

第二范式的规范描述是“如果关系模式 R 是 1NF 且其中的所有非主属性都完全函数依赖于关键字，则称关系 R 是属于第二范式的”，通俗地说，就是要求实体中的所有非标识符属性都必须完全依赖于该实体的标识符属性，所谓“完全依赖”的含义是，根据标识符属性值可以唯一地确定一个实体的其它非标识符属性的值。

为便于理解，下面让我们在前述例子的基础上进行引申——考虑到只记录作者姓名通常不能满足实际应用需要（比如还需要记录作者的性别、联系方式等信息，或者可能出现重名的情况），初学者可能会将表 8-5 所示的图书作者-信息表改成如表 8-6 所示的形式，并将其中的图书编号、作者编号设为联合标识符（主键），以标识表中的每一行记录。

表 8-6 图书－作者信息表

图书编号	作者编号	作者姓名	作者电话
b001	a001	张三	67804832
b001	a002	李四	82567788
b001	a003	王五	67935446
b002	a004	赵六	84153322
b003	a005	麻七	55668822
b003	a001	张三	67804832
…	…	…	…

这样的处理明显会导致一系列的问题：首先是数据冗余，加入一位作者编著了 10 本书，其姓名、电话等信息就将在表中重复保存 10 次；在数据更新时也很麻烦，如果该作者的联系电话出现变更，就必须同时修改表中的 10 条记录，这很容易导致数据的不一致性问题；此外，如果没有书号信息（主键字段值不能为空）、在图书出版之前就无法单独记录作者的信息，只能等图书出版之后才能将作者相关信息存入。

产生上述问题的根本原因是，非标识符属性“作者姓名”、“作者电话”并不是完全依赖于所属实体的组合标识符（“图书编号”和“作者编号”），而只是完全依赖于其中的“作者编号”

（只要根据“作者编号”就可以唯一地确定一位作者的姓名电话等信息），对前者则为部分依赖，即违背了 2NF 的要求。这种问题的解决办法是，将实体中的冗余数据分割出来、并保存到单独的实体中，并在所得到的两个实体间建立引用关系，即将之改造为符合 2NF。

说得具体些，考虑到可能存在多本图书涉及到同一作者的情况，为避免重复记录同一作者的信息（数据冗余），可以创建单独的作者信息表，这样的数据结构就可以比较规范地描述图书信息、作者信息以及二者之间的多对多关系了，此时多对多关系也已被转换为实体（图书－作者信息表）。具体如表 8-7～表 8-9 所示。

表 8-7 图书信息表

图书编号	书 名	出版社	订 价
b001	《Java SE 应用程序设计》	北京理工大学出版社	66.00
b002	《Java Web 应用开发》	科学出版社	55.00
b003	《Java 实用案例教程》	清华大学出版社	23
…	…	…	…

表 8-8 作者信息表

作者编号	姓名	地址	联系电话
a001	张利国	北京市海淀区	010 - 67804832
a002	刘伟	北京市朝阳区	010 - 82567788
…	…	…	…

表 8-9 图书－作者信息表

图书编号	作者编号
b001	a001
b001	a002
b002	a001
b002	a002
b003	a001
…	…

相关概念数据模型如图 8-13 所示。

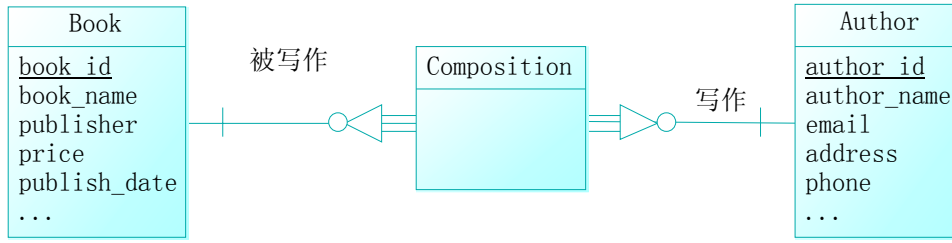

图 8-13 图书-作者概念数据模型

为规范起见，在概念数据模型中将各实体集及属性的命名均改为英文，其中，Composition 实体集对应的是“图书-作者对应关系表”，该表中只有两个外键字段 book_id 和 author_id，他们分别参照图书信息表 Book 中的主键字段 book_id 与作者信息表中的主键字段 author_id，而且，这两个外键字段同时做为 Composition 表的联合主键存在。当然如果需要的话，也可以在 Composition 表中添加更多的字段，比如记录作者的排序信息（是第一作者，还是第二作者等）。

8.4.4 第三范式

第三范式（3NF）在 1NF 和 2NF 的基础上又附加了新的要求，简单地说，就是实体中属性间不得存在传递性依赖关系——非标识符属性必须完全依赖于标识符、且不得依赖于实体中的其它（非标识符）属性。这样的描述读者可能还会感到费解，下面结合一个具体的例子来做说明。

假定企业财务部门需要记录每个员工的员工编号（eid）、基本工资（basic_salary）、岗位津贴（job_allowance）、绩效奖金（performance_bonus）和工资总额（total）等信息，为简明起见，这里没有考虑扣除项目，则可以定义如表 8-10 所示的员工工资表。

表 8-10 员工工资表（Salary）

eid	basic_salary	job_allowance	performance_bonus	total
e001	2500.00	300.00	580.00	3380.00
e002	3478.00	1200.00	1600.00	6278.00
…	…	…	…	…

其中，eid 为主键字段，用于唯一标识表中的每一行记录，同时该字段又做为外键引用了员工信息表中主键字段 eid（员工信息表 empinfo 的定义可参见本书第 7.2.5 节）。

在进一步分析之前，我们需要确认一下 Salary 表是否 1NF 和 2NF 的要求：首先，表中所有字段都是不可分割的数据项、且所有记录行可以由主键字段 eid 进行唯一标识，因此 Salary 表符合 1NF 的要求。其次，表中所有非主键字段均完全依赖于主键字段 eid（根据员工编号可以唯一地确定该员工的基本工资、岗位津贴、绩效奖金和工资总额的数值），由于表中只定义单一字段 eid 做为主键（不存在联合主键），也就不存在对于主键字段的部分依赖问题，因此 Salary 表也符合 2NF 的要求。

接下来考察一下它是否符合 3NF 的要求，问题出现了——表中的 total 字段虽然完全依赖于主键字段 eid，但它同时还依赖于三个非主键字段（basic_salary、job_allowance 和 performance_bonus）

的组合，因为工资总额是将该员工的基本工资、岗位津贴和绩效奖金三者相加而得出来的，这就是所谓的传递性依赖，即 Salary 表违反的 3NF 的要求。实际上，存在传递性依赖关系的 total 字段属于前三个非主键字段的“衍生物”，既然可以通过前三者计算得到，那么从表中将该字段删除也不会对实际应用产生多大影响，且可以避免出现数据的不一致性问题（出现 basic_salary、job_allowance 和 performance_bonus 三个字段值之和与 total 字段值不符的情况）。删除 total 字段之后，Salary 表就符合 3NF 的要求了，需要的话，此时可以通过如下 SQL 语句获取员工的工资总额：

```
SELECT eid, basic_salary + job_allowance + performance_bonus AS total FROM salary;
```

> 说明：在实际应用开发中，如果频繁进行涉及到工资总额的统计查询操作，有的设计者就会保留“工资总额”（total）这一冗余字段、以提高查询统计的效率，这就是“以空间换取时间”了。

归纳一下，数据库规范化的本质就是实现所存储数据的原子性、原始性、演绎性、稳定性和一致性：

- 原子性（Atomicity）

  数据表中的字段应是不可分割的数据项

- 原始性（aboriginality）

数据表中保存的应是原始数据、或者说基础数据，而不应包含通过现有数据能够衍生得到的计算结构（如上文中的 total 字段）。

- 演绎性（syllogistic）

数据表中保存的数据和应用程序代码中的数据（包括程序运行过程中通过其它方式输入的数据），可以演绎/派生出所有业务逻辑所需的结果，即满足应用程序的运行需要。

- 稳定性（stability）

基本表的结构应尽量相对稳定，其中的记录是要长久保存的。

- 一致性（consistency）

冗余数据应保证其取值的一致性，此外，存在引用关系的表间数据也应保证其一致性（参见本书第 7.2.5 节外键约束）。

8.5 数据库设计详细过程与实例分析

数据库设计是数据库应用系统设计的一个组成部分，其核心是针对于特定的应用环境，设计合理的数据模型，创建数据库及其应用系统，使之能够有效地存储和处理数据，以满足用户的应用需求。从实用角度出发，数据库设计可分为如下几个步骤：

第一步：创建概念数据模型

- ◆ 确定实体和关系
- ◆ 确定属性
- ◆ 规范化数据

第二步：生成物理数据模型

第三步：验证设计

为便于学习者理解和掌握，下面结合具体的实例来讲解和展示数据库设计的详细过程。假定我们要开发一个小型的 ERP 系统，以管理公司内部资源，其应用业务场景描述如下：

v512 工作室由 IT 业界专业人士组成，在提供高端 IT 培训业务的同时，还自主制作并免费发布大量公益性学习资源，工作室以公司形式运营，目前共拥有 18 名员工，这些员工分属于 4 个部门，且员工之间存在上下级管理关系。计划将来根据业务的发展设立更多的部门，聘用更多的员工。为保证质量，工作室对其成员的各项专业技能进行了级别评定。

8.5.1 确定实体和关系

确定高级别的活动

要确定本 ERP 系统数据库设计中的实体和实体间关系，首先应明确要基于该数据库执行的高级别活动，这里所谓的高级别活动是指从用户的视角出发，确定本数据库设计中系统所涉及到的业务活动。比如，存储和维护员工的个人信息等。

在前述的应用业务场景中，v512 工作室需要考虑的高级别活动包括：

- 聘用新员工
- 解雇现有员工
- 维护员工的个人信息
- 增设新部门
- 裁撤现有部门
- 维护部门信息
- 维护工作室业务相关的技能信息
- 维护各员工的业务技能掌握情况

确定实体

接下来要确定的是，针对上述的高级别活动需要记录和维护有关哪些事物的信息，这些事物将被转换为实体。其中，员工相关信息可抽象为“Employee”实体、部门相关信息可抽象为“Department”实体、技能相关信息抽象为“Skill”实体，为规范和方便起见，这些实体均采用英文命名，并尽量在名称中体现其含义。

确定关系

进一步对上述高级活动进行分析，以确定实体间存在何种关系。具体包括：

- Employee-Department 实体之间存在隶属关系
  员工必须且只能隶属于某一个特定的部门，一个部门可以包含 0～多名员工，此为一对多关系。这种从两个方向上对同一个关系的细化描述被称为关系的角色，每个关系都对应两种角色。
- Employee-Department 实体之间存在管理关系
  每一名员工可以管理 0～1 各部门，每个部门必须由一名员工负责管理（其管理者不必须隶属于本部门），此为一对一关系。
- Employee-Skill 实体之间存在掌握关系
  每一名员工均应掌握 1～多项业务技能，每项技能可能被 0～多名员工掌握，此为多对多

关系。

- Employee-Employee 实体之间存在管理关系
  每位员工由 0～1 位上级员工负责管理，有的员工可能没有上司（比如公司经理），但有的话只能有一位直接上级。上级员工可以管理 0～多位为下级员工。

经分析而得的上述实体间关系如图 8-14 所示。

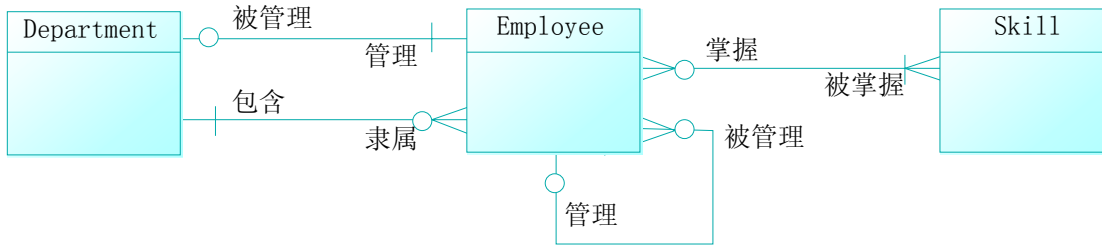

图 8-14 数据库设计实例 CDM 之 1

将多对多关系更改为实体

前文中已讲过，在多对多关系中，可能会出现有属性与关系相关联、而不是单纯的与实体相关联的情况，将这样的属性添加到任何一个实体中都是不合理的，此时应将该关系转换为、或者说定义为实体（详见本书第 8.2.2 节）。我们这里的 Employee 与 Skill 实体之间就存在这种情况——员工所掌握的技能项目及其评定等级信息就与两个实体之间的多对多关系相关联，因此将此多对多关系定义为“人力资源”（HR）实体，转换后的实体-关系如图 8-15 所示。

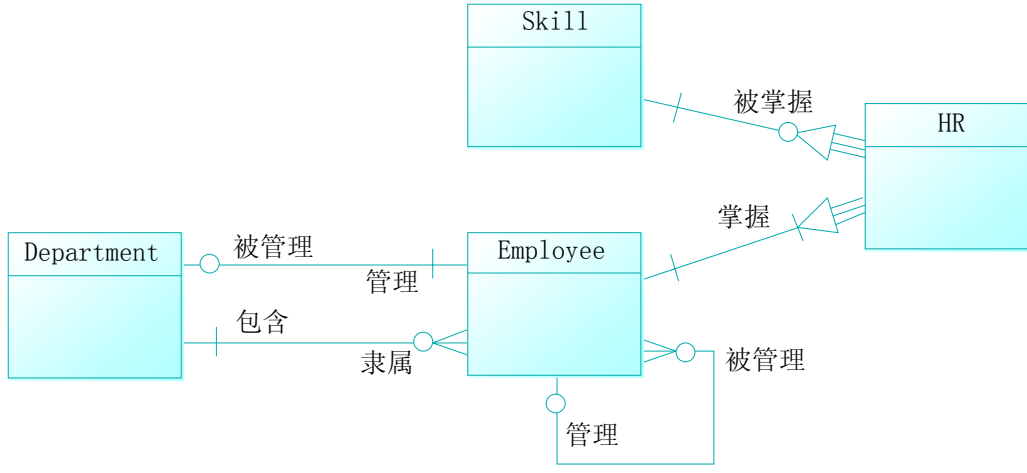

图 8-15 数据库设计实例 CDM 之 2

在实际的数据库应用设计中，更简单而可行的处理原则是，将一切多对多关系均转换为实体，而不必逐个对其进行细化分析，这样可以很好地解决违背数据库规范化第二范式的问题。

分解活动

接下来，需要对前述的高级别业务活动做进一步分析，看是否可以将其中的部分或全部项目分解为相对低级别、或者说更细致的业务活动，比如，其中“维护工作室业务相关的技能信息”

这一高级别的活动可继续分解为：

- 添加新的技能项目
- 删除不再使用的技能项目
- 维护/更新现有技能项目的详细详细

最后一项高级活动“维护各员工的业务技能掌握情况”也可以进行类似的分解。需要说明的是高级业务活动的确定以及这里的细化分解并没有一成不变的标准，这就好比实现某一预期功能的编程工作是没有标准答案的（有的只是参考实现），其目的都是为后续的确定业务规则和实体属性等步骤提供便利，如果业务逻辑不是很复杂的话也可以一步完成，具体由数据库设计者自行把握即可。

确定业务规则

接下来要做的是，对前述业务活动进行再次分析，确定其应遵守的具体规则。比如，一名员工必须且只能隶属于某一个部门就是一个业务规则。业务规则通常可以表示为一对一、一对多和多对多关系，或者相应的约束条件，这些规则将来会体现到数据库的结构之中。本实例中相关的业务规则如下：

- 员工必须隶属于某一个部门
- 员工编号一经确定不得更改
- 员工的姓名、性别、职务、所属部门等其它个人信息可以更改
- 员工所掌握的技能项目及其评定等级可以更改
- 每个部门允许设置 0～多部电话
- …

和前述业务活动的确定及分解情况类似，业务规则的确定以满足实际业务需要为准，注意不要搞得过于繁琐而失去其实用价值，具体详略程度需设计者根据经验自行把握，也可以待后续环节暴露出问题后再追溯回来，进行迭代处理。

8.5.2 确定属性

首先，根据前述分析的业务活动的需要确定所有要记录和维护的数据条目，然后再将这些数据条目做为属性关联到相应的实体或关系中，比如：

- 员工实体
  员工编号、员工姓名、性别、职务、上司工号、工资、出生日期、所属部门
- 部门实体
  部门编号、部门名称、部门经理、部门地址、电话号码
- 技能实体
  技能编号、技能名称
- 人力资源实体
  员工编号、技能编号、掌握程度

设置属性后的实体-关系如图 8-16 所示。

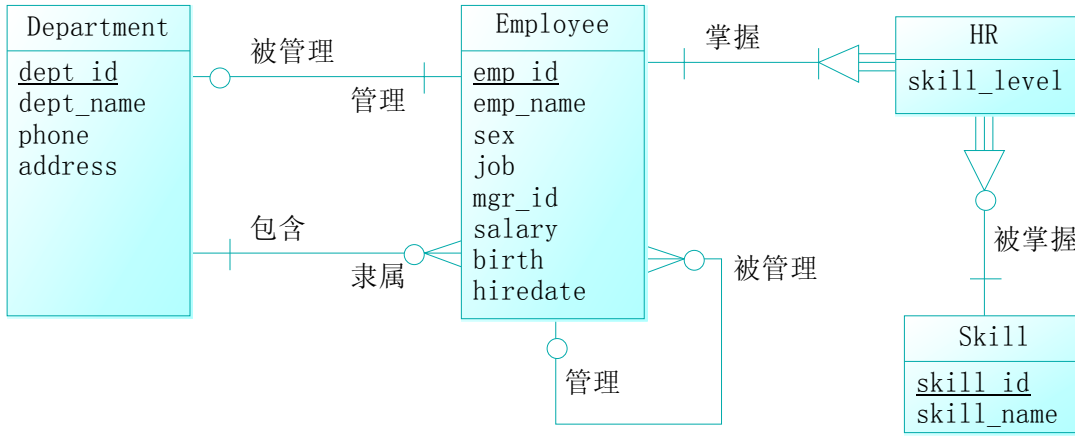

图 8-16 数据库设计实例 CDM 之 3

其中，带有下划线的属性为所在实体的标识符。

为实体或关系设置属性时，应尽量采用规范的命名规则——属性的名称应体现其含义、且应遵循统一的命名格式，以便于将来的使用和维护，比如，假定部门编号使用缩略名称 dept_name，那么部门名称等相应的其它属性就不应该使用完整名称，如 department_name，而应是其名称保持一致，如 dept_name。

在本阶段，设计者只需根据自己的经验和判断进行设计即可，而不必强求数据（属性）与实体间的关联关系一定是正确或合理的， 后续的环节还将对现有的数据模型进行规范化处理，以发现可能存在的问题并进行修正。

8.5.3 规范化数据

前文中已经讲过，规范化数据的目的在于避免出现数据冗余和操作异常，进而节省存储空间并更好地确保数据的一致性（详见本书第 8.4 节），实际上就是对所设计的数据模型进行一系列的规范化测试（判断其是否满足指定的范式要求）、发现问题并进行整改的过程。

在进行数据的规范化测试之前，应先简单地罗列出各实体中的数据，并为每个实体确定一个唯一的标识符、以唯一地标识实体集中的每一个实体，标识符可以是一个属性、也可以由多个属性组合而成。实际上这一工作我们已经完成了，参见第 8.5.2 节的图 8-16，其中，使用下划线标记的即为标识符属性，HR 实体（多对多关系转换而成的实体）的标识符由其所依赖的两个实体的标识符（emp_id 和 skill_id）组合而成。

第一范式测试

根据第一范式的要求，检查各实体的属性是否存在多值的情况，这里的“多值”指的是同一个属性在同一个实体上可以有几个不同的取值，如果有则应将这样的属性从其所属的实体中删除掉，并用这些删除掉的属性创建新的实体和关系。

在我们的设计实例中，假定一个部门可以有 0～多个电话号码，则 Department 实体中的 phone 属性就存在多值的情况，解决办法是，将 phone 属性从 Department 实体中删除，并创建一个单独的 Phone 实体（严格来说是实体集）。此时，Department 实体和 Phone 实体间存在一对多的关联

关系——每个部门可以有 0～多个电话号码，每个电话号码必须也只能隶属于某一个特定的部门。修正后的实体-关系如图 8-17 所示。

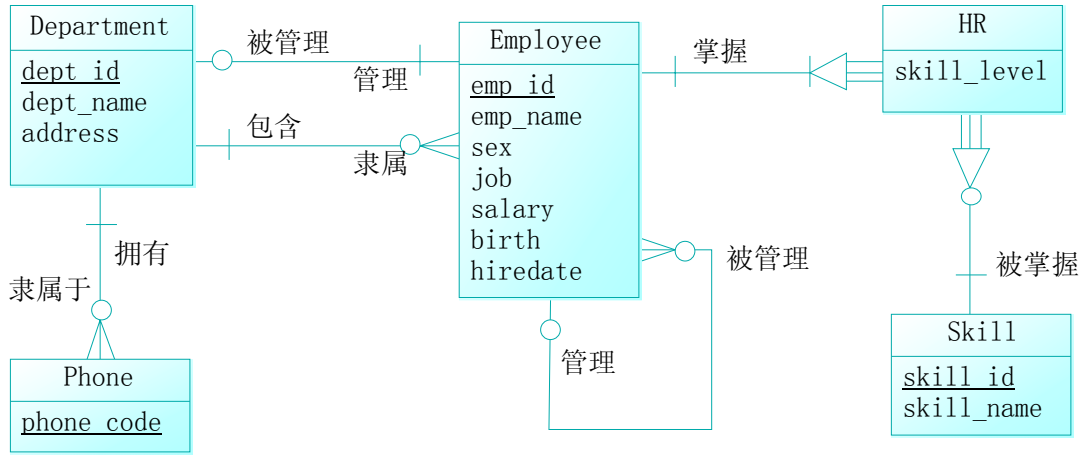

图 8-17 数据库设计实例 CDM 之 4

第二范式测试

第二范式要求各实体中所有非标识符属性都必须完全依赖于实体的标识符，如果存在非标识符属性部分依赖（而不是完全依赖）于该实体的标识符的情况，则应从当前实体中删除这些属性、并将之加入到适合的实体中，其相关原理分析及具体做法前文中已有详细讲解（参见本书第 8.4.3 节）。

实际上，由于单个属性做为标识符的实体中不可能出现非标识符属性对标识符部分依赖的情况，我们只需检查那些由多个属性联合组成标识符的实体。本设计实例中，只有 HR 实体中存在两个属性（emp_id 和 skill_id）组合构成标识符的情况，而其 skill_level 属性的确完全依赖于这二者，故本设计已符合第二范式的要求。

第三范式测试

第二范式要求实体中的属性间不得存在传递依赖，即所有的非标识符属性必须完全依赖于标识符、且不得依赖于实体中的其它（非标识符）属性，如果存在传递性依赖的情况，则应从当前实体中将相关属性删除，有必要的话，也可以将这些属性添加到适合的其它实体中。

经逐个分析，前述各实体的属性之间均不存在传递性依赖，故本设计已符合第三范式的要求。

8.5.4 生成物理数据模型

在确定概念数据模型并完成了数据库的规范化之后，数据库设计就基本完成了，接下来要做的是生成与前述概念数据模型相对应的物理数据模型。这个过程也称作解析关系，因为其主要工作就是将模型中的实体转换为表、并将实体间关系转换为表间的外键关系。具体原则如下：

将实体转换为表

实体中的标识符转换为表的主键，实体中的属性转换为表中的字段，属性的数据类型转换/具体化为特定数据库管理系统所支持的数据类型。

解析关系

概念数据模型中实体间关系最终将被解析/转换为物理数据模型中表的外键，具体可分为如下三种情况：

- 解析一对多关系

在解析一对多关系时，“一”方表中的主键字段（由其对应实体中的标识符转换得来）将成为“多”方表中的外键字段。例如，图 8-18 所示的 CDM 中，Department 实体和 Phone 实体间存在一对多关系——一个部门可以拥有 0～多部电话，但一部电话必须也只能隶属于一个部门。

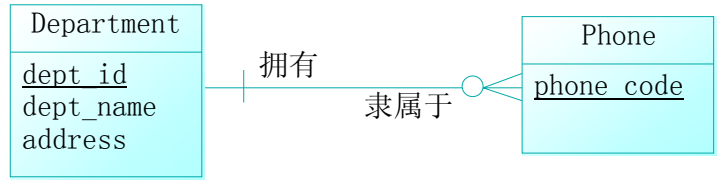

图 8-18 一对多关系 CDM

在转换后的 PDM 中，Department 表中的主键字段（dept_id）成为 Phone 表中的外键字段，具体如图 8-19 所示。

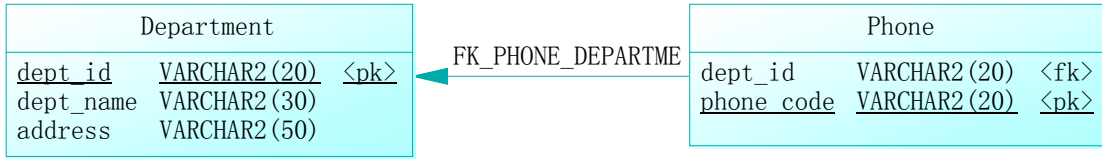

图 8-19 一对多关系 PDM

- 解析一对一关系

在解析一对一关系时，可以在其中任意一方（表中）设置外键。如果该一对一关系在一个方向是强制性的，而在另一方向是可选的，则应在可选的一方设置外键。例如，图 8-20 所示的 CDM 中，Department 实体和 Employee 实体间存在一对一关系——一个部门必须也只能由一名员工（部门经理）负责管理，此为强制性关系；一名员工至多可以管理一个部门，此为非强制性关系。

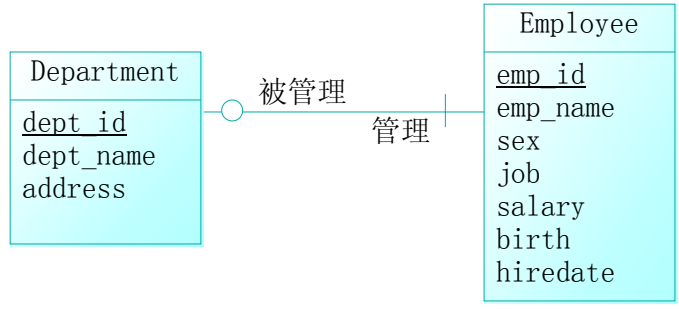

图 8-20 一对一关系 CDM

在转换后的 PDM 中，Employee 表中的主键字段（emp_id）成为 Department 表中的外键字段，具体如图 8-21 所示。

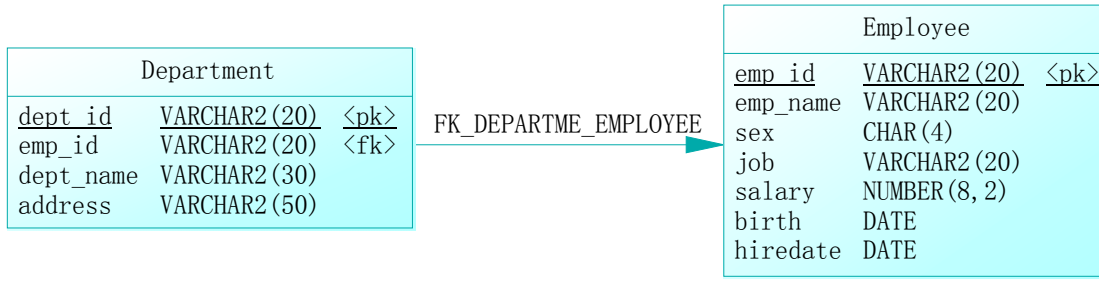

图 8-21 一对多关系 PDM

- 解析多对多关系

对于多对多关系，应将之转换为实体，以便于在数据库中实现，否则很容易出现违背数据库规范化第二范式和第三范式的情况（参见本书第 8.4 节）。例如，图 8-22 所示的 CDM 中，Employee 实体和 Skill 实体间存在多对多关系——每位员工均掌握一门以上的业务技能项目，此为强制性关系；每门业务技能项目可能被 0～多名员工掌握（有的技能项目被为公司业务所需，但尚未招聘到掌握该技能项目的员工），此为非强制性关系。

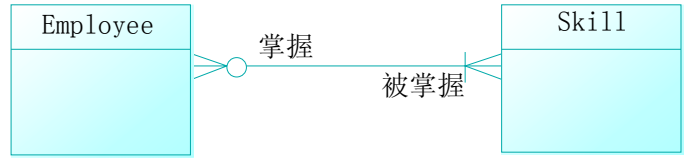

图 8-22 多对多关系 CDM

上述的多对多关系可转换为实体，具体如图 8-23 所示。

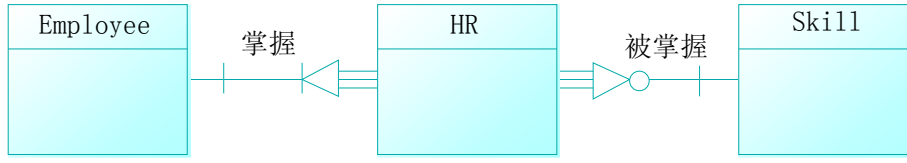

图 8-23 多对多关系转换为实体 CDM

这实际上是将一个多对多关系转换成了两个一对多关系。转换而得的 HR 实体集中的每个实体都必须在 Employee、Skill 实体集中各存在一个对应的实体（两个强制关系），因为员工对技能项目的掌握关系只有在与一个特定的员工和特定的技能项目实体相关联时才有意义。换句话说，新建的 HR 实体既依赖于 Employee 实体、同时又依赖于 Skill 实体，后两者中的标识符组合起来则可以唯一地标识 HR 实体集中的一个实体，HR 实体集方框两侧小圆圈和鸟爪状标记中间的三角形就是用来标明这种依赖性的。

将多对多关系转换为实体后更适合其相关属性信息在数据库中的存储，这样可以很好地避免数据冗余。比如，如果需要记录员工对技能项目的掌握程度，则可以在新实体 HR 中添加相应的属性（skill_level），确定属性后的 CDM 如图 8-24 所示。

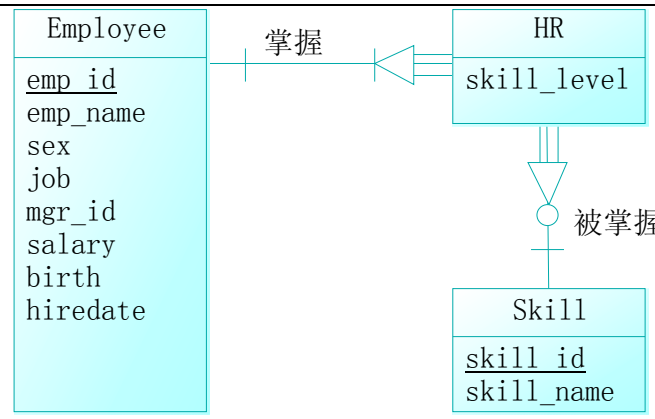

图 8-24 多对多关系转换为实体 CDM 之二

说明，被依赖的 Employee 实体和 Skill 实体中的标识符将自动成为 HR 实体的联合标识符，以唯一标识每一个 HR 实体，由于在当前的 CDM 中已使用三角形标记标明了这种依赖关系，因此不需要 HR 实体集方框中再显式添加 emp_id 和 skill_id 属性。

基于图 8-24 所示的 CDM 而生成的 PDM 如图 8-25 所示。

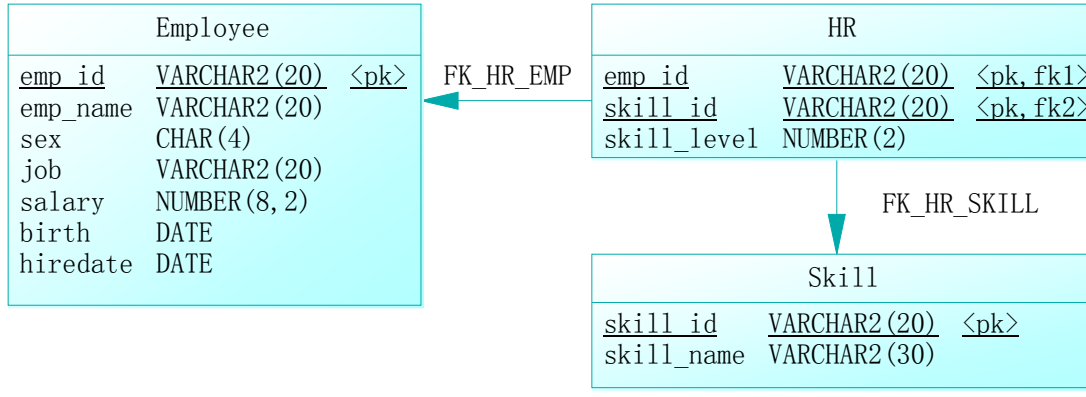

图 8-25 多对多关系转换为实体 PDM

多对多关系转换为实体后，需要再对所得到的新实体和两个一对多关系进行规范化检查，最可取的做法是在创建概念数据模型阶段就将多对多关系转换为实体（参见第 8.2.2 节），以避免到生成物理数据模型阶段进行关系解析时的迭代规范化处理。比如，在本设计实例中，由于在前述步骤（创建概念数据模型->确定实体和关系，参见第 8.5.1 节）中已经将多对多关系转换为实体，因此这里已不存在待解析的多对多关系。

物理数据模型与特定的数据库管理系统相关，比如表中字段的数据类型会随数据库种类和版本号而存在差异，基于 Oracle9i 数据库，图 8-17 所示的概念数据模型所转换成的物理数据模型如图 8-26 所示，本数据库设计满足数据库规范化第三范式的要求。

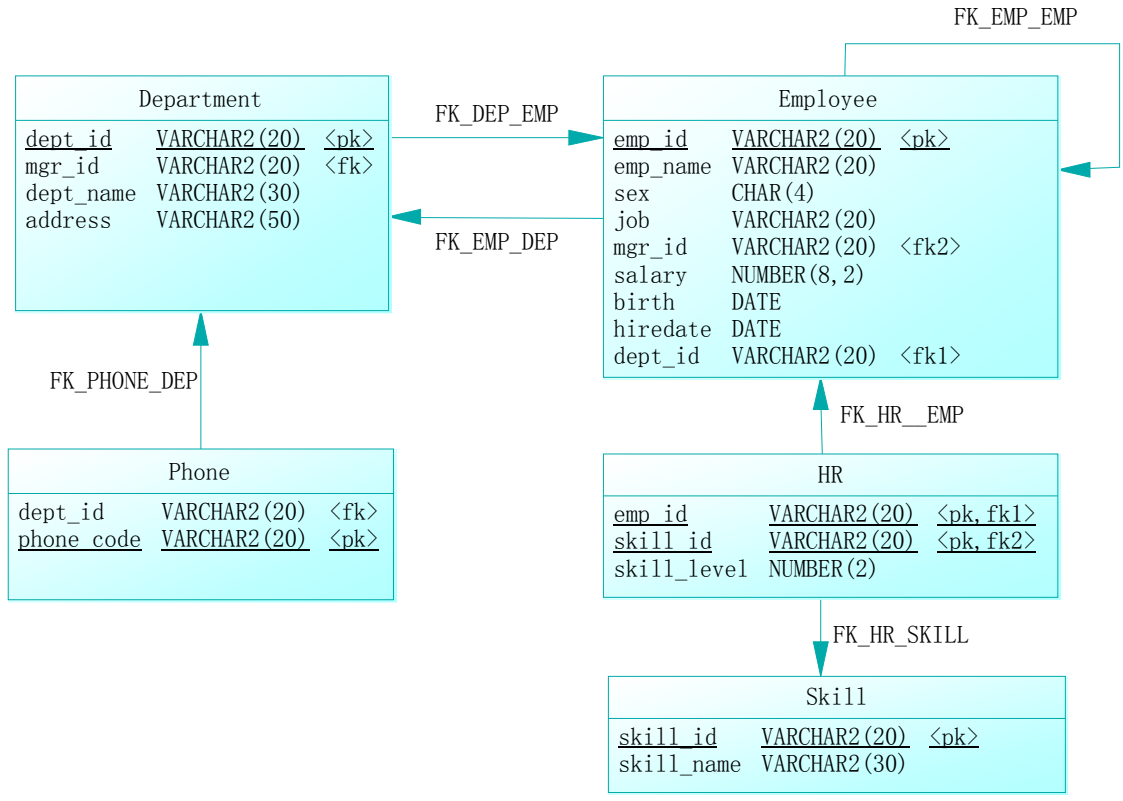

图 8-26 数据库设计实例 PDM

8.5.5 验证设计结果

在实施上述设计结果之前，需要对齐进行最后的验证、以确保该设计能够满足应用开发的需要。具体来说，就是再次检查在设计过程之初确定的各项业务活动（参见第 8.5.1 节），以确保基于当前的数据库设计可以访问和操作业务活动所需要的全部数据。比如，是否可以满足应用开发中聘用新员工、解雇现有员工、维护员工的个人信息等业务活动对数据的访问和操作要求，在本设计中，答案是肯定的。例如：

如果经验没有问题，那么就可以实施本设计结果了。这里所谓的“实施”，具体来说就是根据先前的物理数据模型设计结果，运用目标数据库管理系统（DBMS）所支持的数据语言、客户端或服务器端工具、以及应用开发宿主语言（如 Java），在目标数据库管理系统中创建各种数据库对象，编制与调试应用程序，组织数据入库，并进行调试运行。

基于 Oracle9i 数据库，实施本数据库设计的最终结果（图 8-26 所示的 PDM）的对应 SQL 脚本文件如下：

```
--丢弃已创建数据表（供调试之用）
DROP TABLE hr;
DROP TABLE skill;
DROP TABLE phone;
```

```
DROP TABLE department CASCADE CONSTRAINTS;
DROP TABLE employee;

--创建部门信息表
CREATE TABLE department(
	dept_id VARCHAR2(20) PRIMARY KEY,
	mgr_id VARCHAR2(20),
	dept_name VARCHAR2(30),
	address VARCHAR2(50)
);

--创建员工信息表
CREATE TABLE employee(
	emp_id VARCHAR2(20) PRIMARY KEY,
	emp_name VARCHAR2(20),
	sex char(4),
	job VARCHAR2(20),
	mgr_id VARCHAR2(20),
	salary NUMBER(8,2),
	birth DATE,
	hiredate DATE,
	dept_id VARCHAR2(20)
);

--创建部门电话信息表
CREATE TABLE phone(
	phone_code VARCHAR2(20) PRIMARY KEY,
	dept_id VARCHAR2(20)
);

--创建技能项目信息表
CREATE TABLE skill(
	skill_id VARCHAR2(20) PRIMARY KEY,
	skill_name VARCHAR2(30)
);

--创建员工技能等级评定信息表
CREATE TABLE hr(
	emp_id VARCHAR2(20) ,
```

```
    skill_id VARCHAR2(20) ,
    skill_level NUMBER(2),
    CONSTRAINT hr_pk PRIMARY KEY(emp_id, skill_id)
);

--添加外键约束
ALTER TABLE hr ADD CONSTRAINTS hr_emp_id_fk
    FOREIGN KEY(emp_id) REFERENCES employee(emp_id);

ALTER TABLE hr ADD CONSTRAINTS hr_skill_id_fk
    FOREIGN KEY(skill_id) REFERENCES skill(skill_id);

ALTER TABLE phone ADD CONSTRAINTS phone_dept_id_fk
    FOREIGN KEY(dept_id) REFERENCES department(dept_id);

ALTER TABLE department ADD CONSTRAINTS department_mgr_id_fk
    FOREIGN KEY(mgr_id) REFERENCES employee(emp_id);

ALTER TABLE employee ADD CONSTRAINTS hr_dept_id_fk
    FOREIGN KEY(dept_id) REFERENCES department(dept_id);

ALTER TABLE employee ADD CONSTRAINTS hr_mgr_id_fk
    FOREIGN KEY(mgr_id) REFERENCES employee(emp_id);
```

联系作者：

张利国 zhangliguo@tsinghua.org.cn

相关资源：

新科海学校 http://www.jobedu.com.cn/
v512 工作室 http://www.v512.com
高清视频下载 http://www.verycd.com/topics/215898/