

## 第2章 逻辑数据库的设计和标准化

本章要点：

实体-关系模型

将实体关系图映射为关系模型

理解标准化

### 2.1 实体-关系模型

对一个数据库管理员来说，所能为他的数据库做的最好的事情就是使之开始于一个合理的逻辑设计。不幸的很，数据库设计常常被匆匆地完成以致于做错，甚至在数据库建立后重新返工。一个见闻广博的和聪明的数据库管理员知道对数据库进行很好的设计，会大大提高数据库的性能，而不是减损数据库的性能，这种思想与流行的思想相反。事实上，直接投入物理设计或更深层的工作，只会带来麻烦，不仅在性能方面，而且在数据完整性方面同样如此。如果一个数据库运行得很快，但收藏的数据却是错误的，这又有什么好处呢？而且，在数据库系统的早期设计阶段，创建一个合理的逻辑设计，可以让它接受以后创建和维护阶段物理设计改变的考验。可是，如果你在逻辑设计阶段走捷径，你将不但可能需要重新设计逻辑模型，而且还可能需要重新构造下面的物理模型。间接的代价（职员的工作时间、停工期等等）可能会是令人吃惊的。在进行和建立数据库之前，需要了解逻辑数据库设计和标准化背后的基本原则。

在70年代中期，关系数据库模型逐渐超越其他的数据模型占据主导地位，关系模型技术的风靡使设计性能得到规范化。这其中最流行的是实体关系图（Entity-Relationship Diagram, ERD），它是P.P.Chen在1976年提出来的。这就是语义数据模型，因为它试图捕获业务要素（业务本质）的语义或正确含义。因为关系模型本身几乎就是一个依据语法的模型，是一种主要处理结构的模型，实体关系图（ERD）通常用于补充它。实际上，ERD建模必然先于关系建模。当一个ERD结束时，它或多或少地被直接映射到关系模型上，而后关系模型再被映射到它的物理模型上。

一个实体是一个业务元素，比如一个雇员或一个项目。一个关系就是两个实体之间的联系，比如工作于不同项目的雇员。属性即组成实体的特征，比如一个雇员的工资或项目的预算。属性被认为是来自定义域中的取值或值的集合，它们所取的值是它们以后在关系模型中所用到的数据。它们是对一个事物全部抽取或部分抽取。ERD有许多画法，只要你选择一种并在整个使用过程中保持含义一致即可。

使用方框代表实体画高级图（那些不带属性的），将实体的名字列于方框的中心。低级图的实体名称列于方框中的上部，后面跟着属性名称。在方框之间画有箭头，代表关系类型，有三种基本类型的关系：一对一、一对多以及多对多。一对一的关系根据一对一关系的类型，在线条的一端或两端使用单箭头。一对多使用双箭头，多对多在两边使用双箭头。当一个实体的每一个值都和另一个实体的一个值并且只有一个值有关时，就存在着一个纯粹的一对一

关系，反之亦然。这种类型的关系是很少见。图 2-1 展示一种一对一关系，一个丈夫只能和一个妻子结婚，而且一个妻子也只能和一个丈夫结婚（没有考虑一夫多妻或一妻多夫的情况）。

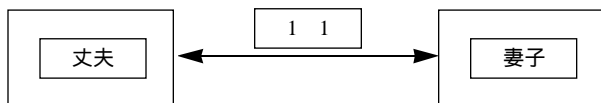


图2-1 一对一（1:1）关系

一种更为普遍的一对一关系是子类型关系，这是面向对象分析和设计的基础之一。在面向对象系统中，这被看作是类和子类（或者更简单地说，类的级别）。图2-2显示一个一对一子类型关系是如何被模拟的，该图显示了一个经典的例子：正方形是长方形的子类型。箭头的方向指明了继承的方向，继承是有关类级别的另一个面向对象概念。换句话说，在更为普遍的实体中的属性（长方形）上，将属性（如长和宽）送给更为特定的实体（正方形）。因此，继承的方向是从一般到特殊。

子类型关系比纯类型的一对一关系更为常见，但这两种都不常用。通常，当一个设计者偶然遇到一对一关系时，他必须问下列问题：

这两个实体能结合吗？

它们对于自己的目标是否是完全相同的？

它们是否由于某些业务原因必须保持独立和不同吗？

通常情况下，一对一实体是可以合并的。

在关系模型中，使用得最多的关系是一对多关系，图 2-3 显示了一种一对多关系。一个州有许多城市，但所有这些城市只属于一个州（这是正确的。然而，你会发现一个城市的名字被不同的州重复使用，这只代表着一件事：作为一名设计者，你选择的主键一定不能是一个城市的名字，例如：主键可以是州名+城市名。前面的章节包含了主键的定义）。

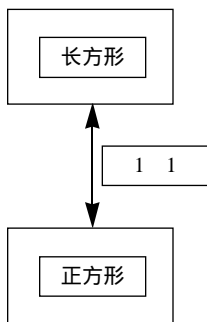


图2-2 一个一对一（1:1）的子类型关系

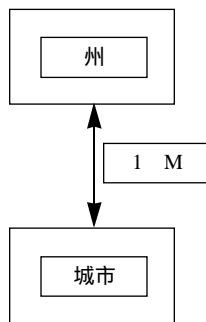


图2-3 一对多（1:M）关系

最后，图2-4表示许多雇员正在从事多个项目——这是一个多对多关系。注意带有下划线的属性是标识符属性，代表着你当前推测它们有可能最后成为关系模型中的主键。

在这种情况下，作为一名设计者，你所能为自己做的最好的事情就是把自己从所有的多对多关系中解脱出来；并不是真正地除去它们，但你可以在它们原先位置上使用两个或多个一对多关系来代替多对多关系。之所以要这样做，是因为关系模型并不能实际处理一个多对多关系的直接实现。仔细想想，如果有许多从事多个项目的雇员，你怎样来贮存外键？你不能在一列中贮存多个值，这样违反了数据必须是原子的关系型要求，这意味着没有一个单元

能够持有一条以上的信息。信息法则（在第1章已讨论过）的这一示例也说明：它是第一范式（First Normal Form）的一个特殊情况。稍后我们将讨论第一范式。因此，为确保数据的原子性，每条多对多关系都被两个或者多个一对多关系所取代。

因此，你所要作的工作就是分割多对多关系，这样多名正从事多个项目的雇员就变成了一个雇员拥有多项职位和一个项目分配给多个职位，而职位成为新的实体。图2-5表示了这种新的关系，注意，标识符属性已经被合并。

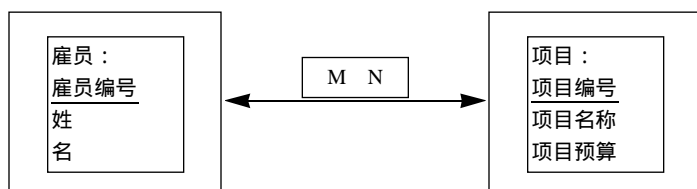


图2-4 多对多（M:N）关系

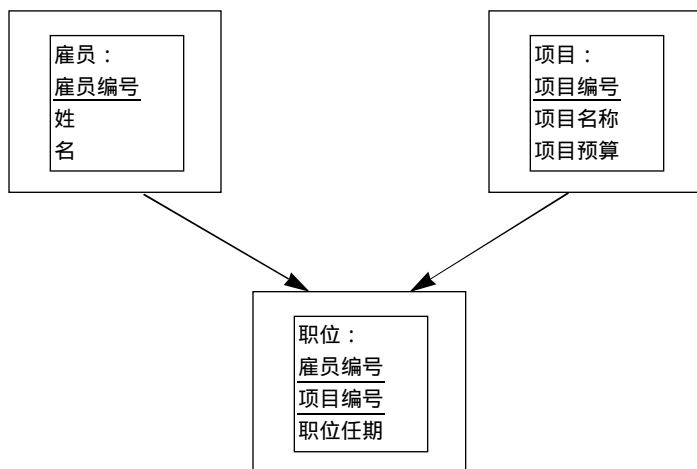


图2-5 使用两个一对多（1:M和1:N）关系修改多对多（M:N）关系

在关系模型中，被称为职位的新实体通常被叫做交叉表，因为它代表着与其相关的两个表中每一对实际值的交集，有时也称为叫纽带表（junction table）或连接表（join table），交叉表是这样的一个实体：它不一定总是一些业务元素的真实抽象，但是它是解决和实现关系模型中多对多关系的基本方法。

## 2.2 将实体关系图映射为关系模型

一个ERD可以准确地映射到关系模型上，因为它就是为这个目的而创建的。实质上，实体变成了表，属性变成了列，标识符属性变成了主键。如果不通过交叉表，关系并不能真正实现。外键是这样创建的：把一个单方表中的主键放入一个多方表中。例如：一个州对应许多城市的关系要求你把州的主键放到城市表中，在那里创建一个外键，这样在两者之间就建立了关系。

当前市场上有许多自动化计算机辅助软件工程（Computer Assisted Software Engineering，CASE）工具能够帮助你实现这个映射。这样的工具包括 LogicWorks公司的ERwin和Oracle公

公司的Designer/2000，这些工具不仅能画出实体关系图，而且能指定主键、外键、索引、约束，甚至能够产生标准数据定义语言（Data Definition Language，DDL）SQL代码帮助你建立表和索引。对于Oracle来说，你可以直接运行这些脚本，但通常需要修改它们，例如修改数据类型或增加存储参数。这些工具还能帮你从一个没有文档的现存数据库反推出该数据库的逻辑模型！当想要集成数据库或承担一个已建数据库的数据库系统管理员的职责时，这是特别有用的。CASE工具不仅能够帮助你设计和建立数据库系统，而且同样也能帮助你形成文档。

## 2.3 理解标准化

标准化是关系模型的改良或扩展，标准化也是依照第一个关系模型草案并在此基础上采用一定具体方法进行改进的进程，这些具体的改进方法你不久将会学到。和关系模型一样，标准化的基础是数学，它是建立在被称为函数相关性（functional dependency，FD）的概念上的。

尽管没有必要深入研究函数相关的数学，但至少为关系模型定义它是有用的和指导意义的。一列或一组列——Y，被认为是与另列或一组列（X）函数相关；如果赋予X一组给定的值，那么将决定Y的一组唯一值。说Y和X函数相关，就等于说X决定Y，通常写作 $X \rightarrow Y$ 。当然，最明显的例子就是，一个关系表中的主键能够唯一决定表中某一行的值。然而，其他相关也可能存在，只是它们不是主键的结果。标准化的主要目的就是消除关系模型中不是主键造成的所有函数相关。

以下是标准化的三个主要原因，在许多数据库分析和设计的书中经常提到它们：

保持数据完整性。这个原因，也许是最重要的原因，足可以解释为什么要实现标准化。因为数据只被贮存一次，所以它保持正确和一致性。换句话说，不必保存数据的多个备份。否则，相同数据项的各种备份可能不再同步，并且可能最终要求繁重的应用程序控制，因为RDBMS的自动完整性机制不能起到调整作用。许多遗留系统正处于这种状态。建立一个尽可能与应用无关的模型。换句话说，标准化只是增强这样一个概念，即关系模型应该是数据驱动的，而不是程序驱动的。对大部分数据库来说，这意味着改变处理需求时数据库设计能保持稳定和原封不动。应用程序的要求应该和数据库逻辑设计无关（尽管像你稍后将看到的那样，它们对物理数据库的设计很有意义）。

减少存储需求（并且也要不断地为提高查询性能打下基础）。除了外键，全面标准化消除了关系设计中所有冗余的东西。不必要的数据备份同样要求不必要的次级存储需求。另外，存有及可能被搜索的数据越多，需要的总系统时间就越多，因此性能也就越差。

### 2.3.1 使用一个标准化例子

在前面的章节，你跳过了原子数据要求（即信息原则）怎样成为第一范式（1NF）的子集，这部分将再次强调它，并更准确地定义1NF。

#### 1. 第一范式

第一范式（1NF）不包含重复组，这等于说贮存在一个单元里的数据必须是一个单一、简单值，而且不能保留一条以上的信息。为了清晰，信息原则不允许在一个列中有重复组，而1NF额外要求在一行中不能有重复组，不管是重复的列还是列中含有的重复信息都不允许。

表2-1列出了去年人口增长率不低于5%的城市所在的州，因为所有城市的信息存储在一个

重复组中，这个表是不标准的或不满足 1NF 的。首先，你怎么能确切知道城市右边列中的人口和百分比是属于那些城市的？当然，你可以给它们假定一个顺序，但是这违反了列是没有顺序的这一基本关系法则。更糟的是，不得不使用数组，这要求终端用户知道并使用一个物理数据结构，如数组。这肯定产生不出良好的用户界面。

表2-1 拥有人口增长率大于等于5%的城市的州

STATE	ABBREV	SPOP	CITY	LPOP	CPOP	PCTINC
North Carolina	NC	5M	Burlington,	40k	44K	10%
			Raleigh	200K	222K	11%
Vermont	VT	4M	Burlington	60K	67.2K	12%
New York	NY	17M	Albany ,	500k	540K	8%
			New York City ,	14M	14.7M	5%
			White Plains	100K	106K	6%

为让它满足 1NF，把重复组从交叉列移到行中，表 2-2 以 1NF 的形式显示了同样的一个表，该表用 STATE 作为主键。然而，这个表还是有问题。如果要更新或删除州的信息，你必须访问许多行，并要在程序上保证它们的一致性（DBMS 不做这些）。如果要插入城市的信息，你必须同时加入州的信息。如果删除一个州的最后一个城市，那么州的信息也将随之删除等等。问题出在那里呢？过一会儿你就会看到。

表2-2 第一范式形式的州和城市

STATE	ABBREV	SPOP	CITY	LPOP	CPOP	PCTINC
North Carolina	NC	5M	Burlington	40K	44K	10%
North Carolina	NC	5M	Raleigh	200K	222K	11%
Vermont	VT	4M	Burlington	60K	67.2K	12%
New York	NY	17M	Albany	500K	540K	8%
New York	NY	17M	New York City	14M	14.7M	5%
New York	NY	17M	White Plains	100K	106K	6%

为了实现更高的标准化级，需要一个非键列。狭义的非键列定义是：一个不是主键的一部分的列；广义的非键列定义是：不是任何候选键的一部分的列。这种情况下，应选择狭义定义。本质上，一个表的列集合可以看作是由一个主键和剩余项组成，剩余项的任意一部分都是非键列。

## 2. 第二范式

第二范式（2NF）没有不完全相关，每个非键列都依赖于全主键，如果主键是复合键，则包括它所有的列。表 2-2 当前并没有遵循这个准则。城市信息不依赖于州信息，即：所有的城市列（CITY、LPOP、CPOP 和 PCTINC）不依赖州的名字（STATE），因此，你可以把它们拆分为两个表（表 2-3 和表 2-4）。这意味着州和城市尽管有关，但它们是单独的实体，并因此应该是不同的表。

表2-3 第二范式（2NF）形式的州

STATE	ABBREV	SPOP
North Carolina	NC	5M
Vermont	VT	4M
New York	NY	17M

表2-4 第二范式 ( 2NF ) 形式的城市

CITY	ABBREV	LPOP	CPOP	PCTINC
Burlington	NC	40K	44K	10%
Raleigh	NC	200K	222K	11%
Burlington	VT	60K	67.2K	12%
New York City	NY	14M	14.7M	5%
Albany	NY	500K	540K	8%
White Plains	NY	100K	106K	6%

### 3. 第三范式

第三范式 (3NF) 没有传递的相关性。没有依赖于其他非键列的非键列，如果一个表所用的非键列都依赖于键、全键，并且只依赖于键，那么这个表就是 3NF 形式的表。如果消除重复组后，每个非主键列都依赖于键和全键，这是 2NF，只依赖于键则是 3NF。城市表（表 2-4）没通过这种检测，因为 PCTINC（增长率）列依赖于 CPOP（当今人口数）和 LPOP（去年人口数量）。实际上，它是这两个列的函数，这类列叫做衍生列，因为它是从其他已经存在的列中衍生而来的。但是，所有的这些列都是非键列，直接的解决办法是删除 PCTINC 列并且在运行时计算它，如果访问它的频率很高，最好使用一个视图。还有，在你的州表（表 2-3）中，SPOP（州人口数）依赖于 ABBREV（简称），因为这是一个候选键，尽管它不是主键。表 2-5、表 2-6 和表 2-7 显示了解决方案，现在你有了 3 个 3NF 形式的表。

表2-5 第三范式 ( 3NF ) 形式的州

ABBREV	SPOP
NC	5M
VT	4M
NY	17M

表2-6 第三范式 ( 3NF ) 形式的州名

STATE	ABBREV
North Carolina	NC
Vermont	VT
New York	NY

表2-7 第三范式 ( 3NF ) 形式的城市

CITY	ABBREV	LPOP	CPOP
Burlington	NC	40K	44K
Raleigh	NC	200K	222K
Burlington	VT	60K	67.2K
New York City	NY	14M	14.7M
Albany	NY	500K	540K
White Plains	NY	100K	106K

### 2.3.2 继续讨论范式

Boyce Codd 范式 (Boyce Codd Normal Form, BCNF) 包括不反转的不完全相关（有时不太正规地称之为 3 1/2NF），主键和它的任何部分都不依赖一个非键属性。因为你取的是非键

的严格定义，3NF考虑候选键问题，而且你的表总是 BCNF形式的。

还有第四范式和更高级范式。在学术上，标准化理论已超出 BCNF许多级了，通常数据库分析和设计的相关书籍已达到 5NF的高度。4NF处理多值相关（MVD）问题，而5NF处理连接相关（JD）问题。尽管这些范式的理论稍稍超出了本书的范围，但你应该知道如果每个 MVD都是一个FD，那么这个表是4NF形式的表；如果每个JD都是它的关系键的结果，那么这个表是5NF形式的表。

第七、第八范式等在一些论著中有介绍。另外，可供选用的范式例如域键范式（Domain Key Normal Form, DKNF）已经被发展，与当前标准化理论并行，或包含了当今的标准化理论。

建议：至少要为BCNF而努力，然后必要时会得到物理数据库设计补偿，这将引导你进入下一个主题。如果可能，研究第四范式和第五范式，设法通过你的努力达到这些级别的标准。你作为一个数据库系统管理员的目标就是尽可能地提高标准化程度，然而使用尽可能少的实体来实现它们。这是一个挑战，因为，通常标准形式越高，产生的实体越多。