

## 第2章 数据库基础

如果认为对象建模的优势在于编写数据库程序时，不必为关系的设计和规范化而烦恼，那就大错特错了。设计一个好的关系数据库的原则，以及规范化的合理使用，也都应当应用到对象关系模型中。

本书的目的不是为了介绍数据库基础知识，本章我们假定读者是数据库的专业人员或已修过数据库课程，从事过关系数据库的工作，且希望转向面向对象模型。本章将简要回顾关系和面向对象理论的一些要点。

### 2.1 关系数据库理论简要回顾

到底什么是关系数据库？简单地在关系数据库产品（例如 Oracle）中存放一些表，并不是已经建立了一个关系数据库。建立一个关系数据库，要求数据库中的表符合基本的关系原理，并能进行关系操作。2.3节“规范化规则”中介绍了这些概念。

为了说明这些原理，本章将给出若干例子，例中存放在展开文件中的数据不符合规范化规则。我们将解释为什么这些结构是不可取的，并给出等价的关系，以及这些关系是如何改善设计的。

例1：下图2-1EMP表中存放雇员的人事档案数据。

EMP			
EmpNo	Ename	DeptNo	DeptName
1	Al	10	Acct
2	Bel	10	Acct
3	Cil	20	Fin
4	Del	20	Fin
5	Ed	10	Acct

图2-1 EMP关系表

这个结构存在以下问题：

若删除财务部门（FIN）的所有雇员信息，则财务部门现存的全部收据都将被删除。

如果部门名称由FIN改为FINANCE，则属于财务部门的每一条职员记录都需要修改。

根据关系理论，解决的方法是将信息存放到两个表（DEPT表和EMP表）中，结构如图2-2所示。

注意在此结构中，将部门信息和职员信息分别存放到单独的表中，如果要查找 Al所在的部门，可以在EMP表中先找到该部门的编号为 10，然后到DEPT表中确定编号 10对应的部门名称。

有人认为将信息存放在一个表中的效率更高些，但是这样产生的数据库难以操作，需另外编写大量复杂的代码，才能克服由单一数据文件所带来的困难。把相关信息存放在多个表

中的问题，正是关系数据库设计要解决的问题。通常数据库的物理实现是在这些相关联的列（称为外码）上建立索引，上面的例子中，DeptNo就是一个外码，这样做可以大大提高查找所需信息的效率。

例2：下图2-3是一张购货单(PURCH\_ORDER)表。

这样，每一张购货单都将存放在一条记录中，这种结构存在以下问题：

为计算一张购货单的总金额，需要在两个地方将PRICE字段乘上QTY字段的值（ $QTY1*PRICE1+QTY2*PRICE2$ ）。

在购买了第三种货物后如何存放呢？就只好为增加的货物再建一个表了。

如果使用下图2-4的结构，将解决所有这些问题。

#### DEPT

DeptNo	DeptName
--------	----------

10	Acct
----	------

20	Fin
----	-----

#### EMP

EmpNo	Ename	DeptNo
-------	-------	--------

1	Al	10
---	----	----

2	Bel	10
---	-----	----

3	Cil	20
---	-----	----

4	Del	20
---	-----	----

5	Ed	10
---	----	----

图2-2 DEPT表和EMP表的关系表示

购货单

PO_NBR	DATE	ITEM 1	QTY 1	PRICE 1	ITEM2	QTY2	PRICE2
1001	1/1/1999	Ball	3	\$4.00	Bat	3	\$5.00
1002	2/2/1999	Bat	1	\$5.00			
1003	3/3/1999	Glove	2	\$2.00	Ball	2	\$4.00

图2-3 购货单表的结构

#### PURCH ORDER

PO_NBR	DATE
1001	1/1/1999
1002	2/2/1999
1003	3/3/1999

#### PURCH\_ORDER\_DTL

PO_NBR	LINE_NBR	ITEM	QTY	PRICE
1001	1	Ball	3	\$4.00
1001	2	Bat	3	\$5.00
1002	1	Bat	2	\$5.00
1003	1	Glove	2	\$2.00
1003	2	Ball	2	\$4.00

图2-4 改进后的购货单表的结构

利用字段PO\_NBR将这两个表关联，就可以使一个购货单号对应任意多个货物记录。

### 基本术语和概念

在例1和例2中，主要讲了表和字段的设计。在关系数据库的设计中，考虑的是逻辑数据模式，而不是物理数据模式。逻辑数据模式体现的是数据库的设计，而物理数据模式体现的是设计的实现。逻辑设计和物理设计使用的术语是不同的，这可能会在理解上产生混乱。下

面的讨论将使用这些术语，因此，为帮助读者弄清它们的含义，列出如下表格：

逻辑关系	逻辑对象	物理对象
实体	类	表
属性	属性	列、字段
实例	对象	行

为保证叙述的一致性，有必要给出如下定义：

**实体。**主要指组织所关心的某些事物，例如雇员、部门、销售。从理论上讲，实体可以只是一些属性的集合，但这样理解实体没有任何意义，最好能这样理解：实体与现实世界中的一些事物相对应。例如：每一个部门实体的实例都对应着组织中一个特殊部门，人的实体则对应着某个人。关系理论中的实体和实体实例分别对应着对象理论中的类和对象。例如，对于实体部门，可以讨论类部门，而类部门中又包含着对象财务。

**属性。**属性是对象或实体实例中包含的一部分信息，比如部门的名称或雇员的年龄。注意，“属性”一词既用于关系又用于对象。

**主码。**主码是关系概念，是实体的一个属性，用来唯一标识实体的特定实例。例如，职员实体中职员标识号可以作为主码，因为每个职员都有一个唯一的 ID 值。主码必须是唯一的，且该字段不能为空值，主码一旦被赋值就不允许被修改，此项规定是实践上的要求而不是理论上的，因为在关系数据库中，主码被当作指针来使用，改变主码，需要改变所有用该码关联的表中相关记录的值，这可能会影响到成千甚至上亿条记录。

**候选码。**一个实体中经常不止需要一个属性作为其主码。可以指定一个候选码作为主码，而其他可能用作主码的属性指定为候选码。

## 2.2 构造一个简单的关系数据库

这里利用一个简单的例子来说明关系数据库的基本概念，有关这些论题的更详尽的介绍将贯穿于本书。

下列步骤用于创建一个简单的关系数据库：

- 1) 定义实体 DEPARTMENT 和 EMPLOYEE (只包括一些重要的属性)。
- 2) 为实体确定合适的主码：

实体	主码
DEPT	DeptNo
EMP	EmpNo

- 3) 为每个实体确定合适的属性：

实体	属性
DEPT	Name, Location
EMP	Name, Age, Gender

- 4) 确定这两个实体之间存在的关系。注意，这些关系将最终转化成称为外码 (FK) 的重叠的列。

- 5) 确定每个关系的类型。在关系理论中，关系的类型可以分为一对一、一对多和多对多关系。

6) 通过图式记号的方式组织所有信息，此方式可形象地表示数据库的设计。

## 关系类型

本节将介绍实体之间三种不同类型的关系，举例说明如何使用 ERD和UML表示法制作图表，并展示将这些图表转化为数据库物理设计的方法。

### 1. 一对多关系

图2-5展示了DEPT与EMP实体之间的一对多关系。

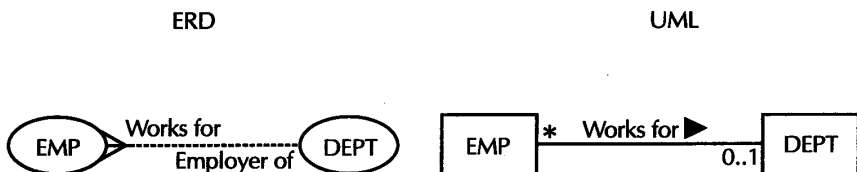


图2-5 一对多关系

下面的两种说法表示了相同的含义：一个部门与任意数量的（包括零个）雇员相关联；一个雇员最多只能与一个部门相关联。在 ERD中，关系两端的命名如图 2-5所示，这样，可以用一对关系语句来描述这个关系，像这种方式：

“每个部门可以雇佣零个或多个雇员。”

“每个雇员可以最多为一个部门工作。”

UML中的符号通常只命名关系一次，并指出名称的相关方向。通常最直观的名称可从“多”的一边读到“1”的一边。图2-2中已显示了如何表示这种关系。可以用更形式化的表示方法，如图2-6所示。

DEPT

DeptNo	DName
PK	CK

10 Acct

20 Fin

EMP

EmpNo	Ename	DeptNo
PK	NN	FK (DEPT)

1 Al 10

2 Bel 10

3 Cil 20

4 Del 20

5 Ed 10

图2-6 图2-2更形式化的表示方法

DEPT表中，“PK”表示主码。“CK”表示候选码，暗示它不可以为空且必须是唯一的。EMP表中，“NN”代表非空。每一个雇员必须有一个名字，但该名字不一定是唯一的。EMP表中的“FK”（外码）表示DeptNo列中的任何值也必须存在于DEPT表的DeptNo列中，而且，如果试图删除它的父部门，则删除操作被禁止或必须自顶向下删除，以防止雇员成为“孤儿”（即指向一个不存在的部门）。这些参照完整性必须被强制执行，不仅在子记录被创建或改变一个部门时，而且在不论何时试图删除父记录时。

下面介绍实现一对多关系的规则。

在关系“1”的一边的表中提取主码，将它作为关系“多”的一边的表的外码。

有时要求每个雇员总在为某个部门工作，在这种情况下，关系就雇员来说时必须的。在 ERD 中，靠近职员实体的线用实线连接，在 UML 中将靠近 DEPT 的 “0..1” 改成 “1”，如图 2-7 所示。

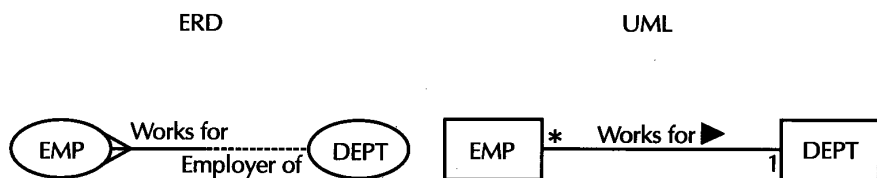


图2-7 一对多关系——“雇员”一边设为强制性的

在关系系统中实现上述情况时，指定 EMP 表中的 FK 不能为空，这样就使所有雇员都与某个部门相联系。

如果要求每个部门都必须至少有一个雇员，可以如图 2-8 来描述这个关系。

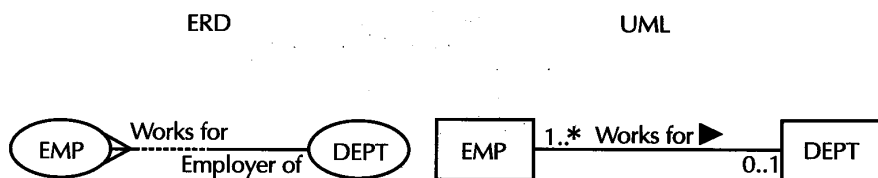


图2-8 一对多关系——“部门”一边设为强制性的

在这种情况下，强制让表执行这一规则并不是一件容易的工作。可以试图使关系的两端都是强制性的，但是，你只能使关系“多”的一边成为强制性的。

总之，一对多关系中，在“1”的一边取得主码，并将它放在“多”的一边作为外码。如果“多”的一边是强制性的，则指定子表的外码非空。为使关系“1”的一边处于强制性状态，必须编写触发器或在应用程序中保证这一点。

## 2. 一对一关系

一对一关系在实体之间不经常使用，实体的每个实例只能与另一个实体的最多一个实例相关联。为说明这个概念，可使用 EMP 和 DEPT 的“管理”关系，如图 2-9 所示。

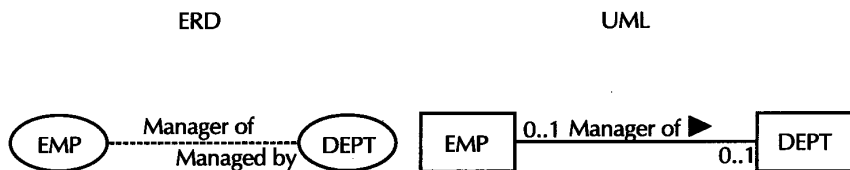


图2-9 一对一关系图

本例中，一个雇员最多只能管理一个部门，而一个部门最多也只能由一个雇员来管理。此结构的关系表如图 2-10 所示。

为联接这两个表，将“FK”置于 DEPT 表中，这是因为部门的数量远远小于雇员的数量，因此，FK 域产生空值的机会将比其放在 EMP 表中小。MGR\_EmpNo 列中，FK 后面的标识符“U”用来表示该列的值必须是唯一的。没有这个唯一性标识符，关系将成为一对多关系，也就是这个唯一性使一对多关系成为一对一关系。

注意没有在这两个表中都存放 FK，规则是将 FK 放于更小的表中。将 FK 放在两个表中是不合理的，这样会产生冗余。关系数据库的一个基本原则是将所有信息存放在数据库中的一

个确切位置，可以像一对多关系中那样定义一对一关系的两端都是强制性的。如果希望每个部门都由某人管理。在 ERD图中DEPT一侧的连线用实线表示，如图2-11所示。

这种情况下，像一对多关系的例子那样，对DEPT表中的FK项加上“非空”标志，如果在关系的雇员一侧加上强制性约束，就无法保证当表的结构发生变化时强制规则的执行。将一对一关系的两端都置为强制性的，意味着这两个实体表示相同的事物而应该被合并，只有当一个实体的属性分成安全属性（例如工资、家庭电话号码）和公用属性（例如姓名、雇佣日期）时，才将一对一关系的两端设置成强制性的。

EMP

EmpNo	Ename
PK	NN
1	Al
2	Bel
3	Cil
4	Del
5	Ed

DEPT

DeptNo	DName	MGR_EmpNo
PK	CK	FK (EMP), U
10	Fin	1
20	Acct	2

图2-10 DEPT和EMP之间的一对一关系表

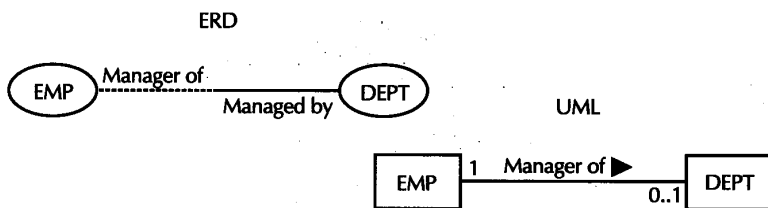


图2-11 一对一关系——“部门”一边设为强制性的

### 3. 多对多关系

实体关系模型中第三种基本关系是多对多关系，如图2-12所示。

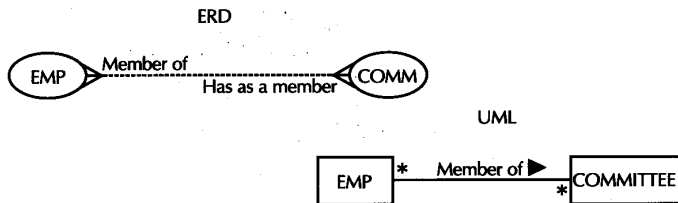


图2-12 多对多关系

这种关系只能在多于两个表时使用。你可能会认为 EMP和COMMITTEE中可以同时设置“FK”。但是，从下面例子中可以清楚认识到这样做是不允许的。例子中包括两个委员会表和两个雇员表，其中每个雇员都服务于两个委员会。解决方法是将关系存放于各自称为“Committee Membership”的表中，如图2-13所示。

本例中，Al、Bel和Cil属于Recreation (REC) 委员会，Bel、Cil和Del属于Employee Relations (ER) 委员会，Ed不属于任何委员会。将FK作为表的主码部分是多对多关系表中比较常见的情况。

无法通过对这些表做简单的修改实现将多对多关系两侧都置为强制性的，必须通过设置触发器或应用程序逻辑来实现。

建立了多对多关系的实体称为交叉实体。UML中称为相关类。交叉实体中的外码类似标

准的一对多关系中的外码，通常可以画出模型来表示。UML为这种类型的结构提供一个特殊的语法。如图2-14所示。

**EMP**

EmpNo	Ename
PK	CK

1	Al
2	Bel
3	Cil
4	Del
5	Ed

**COMMITTEE**

COM_CD	CName
PK	CK

REC	Recreation
ER	Employee Relations

**COMMITTEE MEMBERSHIP**

EmpNo	COM_CD
FK (EMP)	FK (COMMITTEE)
PK	

1	REC
2	REC
3	REC
2	ER
3	ER
4	ER

图2-13 EMP和COMMITTEE之间的多对多关系表

ERD

UML

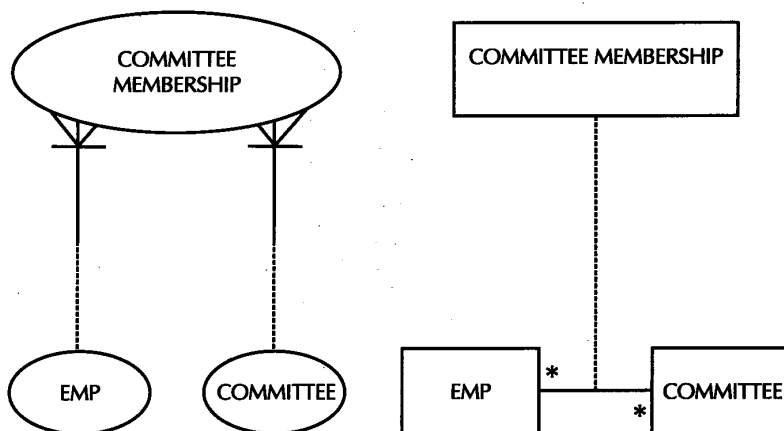


图2-14 EMP与COMMITTEE之间的多对多关系



ERD中的短平线称为 UID线，表明外码在交叉实体中作为主码的一部分。通常采用的办法是删去ER模型中的多对多关系，并仅以交叉实体的一对多关系代替，这样可以使设计者在交叉实体中添加关系，以及为交叉实体创建添加的属性。

## 2.3 规范化规则

一个数据已被规范化，并不是将其转化为“正规的”或“普通的”格式。本书中“规范化”一词来源于数学中规范的概念，意思是正交的。回忆线性代数课程中的规范化一词，它与向量空间的正交向量概念相关联，看看是否一组向量的线性组合能够覆盖一个向量空间。

数据库中的规范化有与其相似的概念，即设计一组数据库表，这组表没有不适当的重叠，并且能像三个正交向量覆盖三维空间那样将所有需要的信息存放到数据库中。这里没有给出规范化的正式定义，针对本书的目的，只对规范化给出实用性定义。

为什么必须建立规范化的数据库？规范化是在与理论的 SEQUEL编程语言结合时产生的。关系理论的基本原则指出，数据库被规范化后，其中的任何数据子集都可以用基本的 SEQUEL操作符获取。这就是规范化的重要性所在。数据库不进行规范化，就必须通过编写大量复杂代码来查询数据。规范化规则在关系建模和关系对象建模中同等重要。

操作数据库中的数据时，仍使用 Oracle的SQL+或SQL语言的对象扩充（SEQUEL的一种实现）来操纵数据库中的信息。不对数据库进行规范化，就无法使用 SQL。一直在Oracle环境工作的人可能意识不到关系数据库以前的环境。在非关系数据库系统中，对特定报表的查找工作至少需要两周时间。对数据库结构的很小修改可能需要开发人员数月甚至数年的时间。对当前系统的体系结构做一些调整的要求往往会遭到拒绝，即使是简单修改。如果在创建面向对象数据库时不使用规范化规则，就会造成系统灵活性的巨大降低。

### 2.3.1 第一范式

第一范式定义数据库中不包含任何多值属性，这里将通过一个不符合第一范式的表结构来介绍。例如，购货单（PO）实体中，假设希望有一个属性名为“Items Ordered”。既然有可能订购多种货物，则实体就应有能力存放这些信息。2.1节例2图中的表是一个不符合第一范式的例子，使用这种结构只允许订购两种货物，在订购三种货物时就出现了问题，这样就需要增加更多的列来存放这些信息。但是，这样又遇上了一个难题，即查找某种货物被订购的次数时，只能通过查找所有列来查找，即使是计算定货单的总金额也需要查找多个位置。

关系模型中有一种方法可实现第一范式，图 2-2给出了满足第一范式后的结构。

不满足第一范式也有可能建立一个好的数据库，关系理论的研究人员很早以前就研究出了用来支持非第一范式数据库扩充的关系理论，这种理论允许数组作为一种数据类型在数据库中使用。这样可以提供比标准关系数据库更高的性能，但需要更复杂的查询语句来检索数据。

利用Oracle 8的对象扩展，有两种方法来实现非第一范式的结构：

数组引用。

嵌套表。

Oracle提供嵌套表和 VARRAY的可查询性，使用这些结构的实现在全书的适当位置将给出



介绍。

2.1节例2中讲到了PO表违反第一范式的不当之处，但是利用 Oracle 8的对象扩充结构来实现非第一范式表也是可能的。

不必对Oracle将数据库扩展为非第一范式的结构而感到不安，数据库界人士已在数年前就开始考虑第一范式是否总是满足需要的。简单地将一个标准关系数据库改造成不符合第一范式的结构，必然会在某些方面付出代价。此外，还存在着这样的问题：在构造表结构时，是否某特定列事实上是一个数组或值或记录呢？回答是肯定的。

很多情况下都非常需要非第一范式的结构。Codd在几年前就发表了关于扩展SQL语言的必要性来支持数组作为数据类型，这需要增加SQL语言的复杂性来支持非第一范式的结构。每一名设计人员是否都决定使用这种结构还是一个问题。这种技术太新，以至于无法确定在什么地方使用这种结构是最合适的，因此需要工具更方便地支持非第一范式结构，并且需要开发人员来更容易地掌握。非第一范式的结构将在第9章中介绍。

### 2.3.2 第二范式

第二范式允许数据库中用多个属性作为主码，这意味着一些属性依赖于部分主码。理解第二范式和第三范式需要理解依赖性的概念。

#### 依赖性

依赖性是一个难以理解的概念，它是一个与数据有关的事物规则的概念。如果属性B依赖于属性A，那么，若知道了A的值，则完全可以找到B的值。这并不是说可以导出B的值，而是逻辑上只能存在一个B的值。例如，在人这个实体中，如果知道某人的唯一标识符，如社会保险号，则可以得到此人的身高、职业、眼睛的颜色等信息，所有这些信息都依赖于这个确认此人的唯一标识符。通过非主码属性如年龄，无法确定此人的身高，从关系数据库的角度来看，身高不依赖于年龄。事实上，这也就意味着主码是实体实例的唯一标识符。因此，在以人为实体来讨论依赖性时，如果已经知道是哪个人，则身高、体重等等就都知道了。主码指示了实体中的某个具体实例。

在包括多个码的表中，如果只依赖主码中的一部分来确定信息，则违反了第二范式。例如，对于Telephone Call实体，要唯一确定某个打来的电话不仅需要源电话号码，而且需要电话打过来的时间。也可以使用其他候选码作为主码，但前面两个属性已经足够了。

在这个实体中假定存在属性“Phone Location”，该结构是否违反第二范式依赖于模型的构造方法。如果电话固定在办公室的桌子上，那么只依赖源电话号码就可以确定它的位置，这样就违反了第二范式的要求。但如果是为移动电话构造模型，因为每一次打电话都从不同的位置，因而也就不会出现违反第二范式的情况。这个例子说明在数据库中要组织正确的模型，需要清楚地理解事物规则。

当错误指定主码时，也会违反第二范式。例如，这里有一个音像商店的例子，可以用一个实体来表示视盘的出租情况，同时指定顾客标识符、视盘标识符和出租日期三个属性作为主码。因为一盘磁带在同一时间不会被出租两次，所以可以用盘标识符和出租日期来唯一确定某一张盘的出租情况。这样，任何与出租情况相关的属性（如收费情况等）只依赖于盘标识符和出租日期。如果主码中还包括顾客标识符，那就违反了第二范式，但这种错误并不是特别严重。

### 2.3.3 第三范式

第三范式指数据库中不能存在传递依赖关系。从实践的角度看，第三范式是指存在一个属性，它所依赖的属性既不是主码也不是候选码。违反第三范式经常产生严重后果，一旦发现出现了违反第三范式的情况，必须予以纠正。违反第三范式意味着数据库设计出现了错误。

2.1节例1中介绍的例子是违反第三范式的典型范例。如果实体或表中不是主码的属性必须依赖于主码或候选码，而不依赖于表中任何其他属性，那肯定是该属性放到了错误的表中或数据模型本身有缺陷。

### 2.3.4 Boyce-Codd 范式

规范化规则并不能帮助建立好的数据模型，它们所提供的是一种测试手段，用来检验所建立的数据模型是否正确。在大学或研究生的计算机理论课程中，规范化通常作为一种算法来讲授，这种算法将不良的结构转变为规范化的数据库。但是，目前还从来没有听说过有谁在实际创建数据库的过程中使用了这种算法。相反，它经常用来创建可能的最好的数据模型和检查其是否违反了规范化规则。在这一点上，有另外一种规范化规则，它有效地将前三规则结合起来，这一规则称为 Boyce-Codd 范式。该规则可不太严格地表述为：“在表中，可以将其中一列或多列指定为主码，也可以指定其他某些列为候选码，表中也存在着其他属性。不考虑候选码，唯一的依赖关系存在于表中每个属性和整个主码之间。”任何其他依赖性的存在都违反了 Boyce-Codd 范式。

Boyce-Codd 范式是用来考虑规范化的最简单的方法。建立数据模型时，用 Boyce-Codd 范式作为标准来评价依赖性，如果发现了违反该范式的现象，首先识别出是违反了第一还是第二、三范式，这样做主要是为了通知其他开发人员出现了违规现象，相信这样考虑规范化是一种更加直观的方式。

## 2.4 基本的对象理论

对象理论是一个很大的话题。我们假定读者已熟悉基本的对象理论，如果不熟悉这些概念，建议阅读任何一本面向对象编程的书籍。这里将对基本术语做一简要回顾，这些术语对开发面向关系数据库非常重要。

如上所述，对象和类的关系正如关系理论中实体示例与实体的关系。类中的对象与物理设计中表中的行密切对应，因此，对象和类的基本概念对从事关系开发的人员来说就比较好理解了。当然，二者之间存在着很大的区别。例如，类中包括方法（用来控制类行为的代码，将在第13章中讨论）。但是类的基本思想与关系实体是相同的。

封装是对象理论的基本概念，如果为类定义了所有接口方法，并且把对类的存取限制于只能使用这些方法，那么就可称此类已被封装，这将意味着应用程序不能直接存取表。例如，代替直接将记录插入表，应用程序必须先调用一个插入方法，以便来处理实际的表插入操作。

继承类似于关系的子类型，一个子类可以从父类继承属性和方法。

市场上出现了支持永久对象的纯对象数据库产品，这些产品完全支持封装和继承，其中一些使用扩展的 SQL 语言存取数据，但很少有支持服务器端对象查询功能的产品。这些产品引起人们的极大关注，但目前还没有利用这些产品开发大型面向对象数据库的应用。