

第6章 数据类型的域

随着Oracle Designer的使用，域在Oracle界开始流行起来。Oracle Designer允许数据库设计者指定预先设计好的可用于数据库列的参数集，这些可设置的参数见表 6-1。

表6-1 最初的Oracle Designer参数

Datatype	标准Oracle数据类型、 VARCHAR2、 char、 number等
Length	字段的最大长度
Precision	数值字段中小数点后数字的最大位数
Default	插入的数据有 NULL值时所使用的缺省值
Low、 high	数值字段和字符字段所允许的最小、 最大值
Values	字段的精确值

在Oracle Designer中，参数可在逻辑层设置，也可在物理层设置。另外，设计者可以设置平均长度，这样Oracle Designer可精确地估算出表所占用的空间。

Oracle Designer用户在域的使用方面有很大差别。一些用户从来不使用域，而有的用户给每个列都建一个不同的域。我们提倡建一些域，对数据模型中的每一个属性都使用一个域。采用这种方法可以管理数据类型的种类，这使得应用程序的创建变得较为容易，因为只提供几种不同的字段长度。

在Oracle 8对象出现以前，数据库中还没有指定域的方法。现在我们可以采用 Oracle的对象扩展来给列分配子类型。可指定的参数见表 6-2。

表6-2 Oracle 8 参数

Datatype	标准的Oracle数据类型、 VARCHAR2、 char、 number等
Length	字段的最大长度
Precision	数值字段中小数点后数字的最大位数

6.1 域的创建

当设计数据库的时候，应避免产生太多不同的域。如果设计的数据类型少，编码人员就能较容易地建立应用程序。从 GUI设计的角度来看，因为屏幕空间很有限，数据类型少会使屏幕的设计较为简单。限制数据类型的方法就是使这些数据类型基于相对少的域。

域是变量的数据参数的描述，但是，也可以把触发器、验证以及其他信息作为域定义的一部分包括进来。

给设计者的一个提醒就是不要使域过于受限制，域越受限制，数据模型就越不稳定。由此可知限制域是有代价的。要使域同你所理解的业务惯例完全相符，那么随着惯例的改变，数据库结构也不得不修改。例如，让我们为零售商设计了一个销售明细（ Sale Detail ）表，商店中没有一种商品的价格超过 100美元，于是我们就限制“ 销售总额（ Sale amount ）”字段的最大值为100美元。如果商店出现一项超过 100美元的商品时，则这个字段就不得不修改。

第二个例子，我们规定销售总额必须大于 0，这看起来是合理的，但是却使我们无法用同

一个表存储销售利润（销售利润可能会是一个负值）。数据的很多限制应放在应用程序中而不是数据库中，第8章中的或然性约束更是这样。

总之，根据对系统未来需求的理解，使域尽可能地受到限制。这需要一个仔细的权衡，即要实现尽可能多的与数据有关的业务惯例，同时又要考虑将来需求的改变，避免做放松约束的额外工作。

在指定域时有几种观点：

- 1) 几乎不使用域，只对一些特定类型的字段如布尔字段 (Y/N) 才使用域。
- 2) 在数据模型中，对每一个属性使用一个独立的域。该方法不是很有用，因为这样做只有一个作用，即当一个字段改变是数据类型时，若该字段是主码字段或用户正保持独立的逻辑和物理模型的话，改变的数据类型会用外码传播。
- 3) 使用仅支持几种对象的少量的域。采用这种策略，总共可有 10~15 个域。
- 4) 值得推荐的方法是拥有一组相对大而且丰富的域来支持一组详细的、与数据相关的业务惯例。这些域不仅规定简单的字段长度信息，而且规定相对有意义的验证信息。采用最后这一种方法，数据库中的任何一个属性都从相对丰富的域列表中分配一个域。

需要确定要建的域的多样化的程度，这是一个风格问题。如果在一个小的系统上工作，很少的域就足够了。但是，要注意的是人们很少建立一个孤立的系统。应设置域来支持整个的开发环境而不是仅支持一个特定的系统，同时要考虑系统的将来变化，这样数据模型和开发环境都能保持相对稳定。我们已用不足 10 个域建立了系统，但是现在的开发环境已增加到大约 50 个域。本章将介绍整个域环境，并讨论选择域的标准。如果你是首次使用域，我们建议从相对有限的域集（10~20）开始创建一个工程，并了解选择的域是否符合数据库设计的初衷，接着再精心构造域列表，以便最好地符合你的开发风格。

6.2 域类型

域用来提供属性的特征，在数据库中支持数据类型、长度和数值精度（需要时）。即使对这些参数，我们仍想为特定的数据库设置合法的数据类型和长度。数据库间域的一致性既简化了数据库，又简化了应用程序的维护和编写。

在数据库中，应有相对少的不同数据对象类型。对应用程序而言，甚至像规定几种不同的数据长度这样的细节问题也能使屏幕和报表设计极大地简化。但是，又必须注意不要让域过于受限。使用一般的建模风格，如果我们在同一个表中存储不止一个对象类型，在某种程度上放松域以适应不同类型的对象是很有必要的。比如，对存储产品销售信息的类而言，必须限制销售额和销售数量为正数，但是，如果想使用同一个类以支持其他事务类型，如销售利润，销售额和销售数量也可以是负数。同样，一个特定的属性在某个表中强制性地用于一种对象类型，而在另一个表中却是可选的甚至根本不可用。这就意味着数据库约束必须允许字段是可选的。幸运的是，在 Oracle 中约束的放松、去除或屏蔽都得到很好的支持。另一方面，在产品系统中增强约束通常是昂贵和费时的。

本章要讨论域的每一种类型，同时提供例子和建议，以便在建立系统时正确地使用域。

6.2.1 字符约束

任何数据库中的大部分数据都是非数值的。在字符信息上可以设置很多有用的限制。在

数据库中可有定长的字符字段（CHAR）或变长的数据字段（VARCHAR2）。CHAR比VARCHAR2使用的存储空间少，但是，若严格根据摩尔法则（大约每过18个月，计算机的速度和存储容量就要翻一番）来考虑磁盘存储的代价，存储一个字节就不再是相关的考虑因素了。因此，很多设计者都采用给文本字段分配VARCHAR2类型的方法。我们认为这是一个错误。如果一个数据库的业务规则规定文本字段总是有固定长度，那么不执行这个规则本身就是一个错误。例如，美国的社会保障号码是固定的9位数，此时，字段应为CHAR(9)。

数值型的ID字段如社会安全号码，头一位的“0”是同整个号码相关的，这时，该字段就要用字符字段而不是用数值字段来存储。同样，电话号码、员工ID号、合同号以及贷款号等应采用字符字段来存储。

提示 我们使用的规则是：如果数字是用来作为计算和度量的话，就用数值方式存储；如果是作为标识，就用字符字段来存储。

1. 定长的CHAR域

大部分定长的字符域仅有几个字符（少于5个）。但有时候，CHAR域包含较多字符是合适的。当信息需要从一个系统传输到另一个拥有定长CHAR域的系统时，这一点是很有用的。我们开发的一个系统包含政府合同号，为了能够为下游系统所接受，这些合同号正好有12个字符，采用CHAR(12)数据类型可实现这个业务惯例。

对特定属性，经常需要很多CHAR域，这些属性在一特殊系统中可能仅使用一次，但是也可能跨系统使用。例如，可能有一个域用于ZIP代码（5个字符）、ZIP+4（9个字符）和社会保障号码（9个字符）。在ZIP+4和社会安全号码里存储连字符是毫无道理的。我们建议为ZIP+4和社会保障号码创建两个独立的域，即使它们都是用9位字符表示的数字。我们也可以创建一个称为“长度为九个字符的数字代码”的域，但是因为ZIP代码和社会安全号码要经常使用，因此还是为它们创建各自的域比较有意义。另外，假设ZIP代码和社会安全号码在同一个域中，不管其中哪一个在格式上需要改变，从一个单独的域中把这两个分离出来都要费很大的劲。

可以选择在字符字段（如社会保障号码、电话号码）中存储连字号和括号，也可用一个显示格式掩码来显示这些字段。把连字号和括号存储到数据库中的好处是，当用SQL*Plus进行简单查询时数据容易读，缺点是在数据库中有一些浪费的空间。我们选择在数据库中不保存格式信息。但这只是一个如何保存这些字段的偏爱问题。

所有这些例子都可以用最小值和最大值（000...和999...）较为容易地实现，这样可有效地避免字符条目。但是，在一些域中，这种简单的方法不能使用。加拿大邮政编码是字母和六个数字的混排，这只能用触发器或把字母和数字存储到不同字段的方法来实现。

不幸的是，Oracle Designer对域而言不明确支持触发器。为在Oracle Designer中能实现这个结构，可在用户扩展中编写触发器代码，或者在属性描述中明确标识；另外可编写API工具，把触发器应用到基于域的每个属性上。

2. 布尔字段

对布尔属性，使用Y/N表示是/否和真/假，所以总是用后缀_YN来命名布尔属性（详见第5章）。对布尔字段有三个域：

强制的——缺省为是或真。

强制的——缺省为否或假。

可选的——没有缺省值。

在指示器值中要包含一个明确的空值，因为查找一个特定的值比检查一列是否为空更为方便。对可选的布尔属性来说，可使用一单字符CHAR字段，该字段有三个可能的值（Y、N、X）。

用CHAR域明确地表示属性所有可能值也是可能的。最常见的例子就是布尔字段。从逻辑上讲，布尔字段允许三个值：T、F和未知。我们对所有有两个值或两个值外加空值的属性使用布尔变量，如性别。在 Oracle 8 中布尔型不是合法的数据类型，因此用数据类型 CHAR(1) 来存储，其中分别用有效值 T 和 F 来表示真和假。

布尔字段是使用一组命名的属性值的唯一方法。我们认为使用检查约束来支持小的值列表是不好的想法，因为大部分值的列表（无论在分析过程中多么明显地稳定）在生产过程中经常变得不太稳定。从性能的角度看，将一组合法值放到第 7 章讨论的值列表类中的代价很低，因此在各种情况下这么做都是合理的。例如，状态属性经常引起关于合法值列表使用的争论，可能的状态值包括初始化（Initial）、挂起（Pending）、完成（Completed）和取消（Canceled），有些人想进行一下修改，比如增加“批准（Approved）”。Oracle Designer 允许在一个大的参照代码表中存储命名的值列表，我们不提倡这种方法，原因在第 7 章中讨论。

我们推荐在系统中包括表 6-3 和代码 6-1 所示的 CHAR 域，它们可用下列代码生成。在本章稍后部分我们将给出怎样在 Oracle 8i 的表中实现域。

代码 6-1

```
CREATE OR REPLACE TYPE CHAR1 AS OBJECT (  
COL          CHAR (1))  
/  
  
CREATE OR REPLACE TYPE CHAR2 AS OBJECT (  
COL          CHAR (2))  
/  
  
CREATE OR REPLACE TYPE CHAR3 AS OBJECT (  
COL          CHAR (3))  
/  
  
CREATE OR REPLACE TYPE CHAR5 AS OBJECT (  
COL          CHAR (5))  
/  
  
CREATE OR REPLACE TYPE SSN AS OBJECT (  
COL          CHAR (9))  
/  
CREATE OR REPLACE TYPE ZIP AS OBJECT (  
COL          CHAR (5))  
/  
  
CREATE OR REPLACE TYPE ZIP_4 AS OBJECT (  
COL          CHAR (9))  
/  
  
CREATE OR REPLACE TYPE CODE AS OBJECT (  
COL          CHAR (10))  
/  

```

表6-3 推荐的CHAR域

名字	类型	长度	精确度	Min	Max	值	缺省值
Char1	Char	1					
Char2	Char	2					
Char3	Char	3					
Char5	Char	5					
ZIP	Char	5		00000	99999		
ZIP+4	Char	9		000000000	999999999		
SSN	Char	9		000000000	999999999		
Bool Yes Opt.	Char	1				Y, N, X	Y
Bool No Opt.	Char	1				Y, N, X	N
Bool Opt	Char	1				Y, N, X	X
Bool Yes	Char	1				Y, N	Y
Bool No	Char	1				Y, N	N
Code	Char	10					

单列的、用户自定义的数据类型可以用它们的名字进行引用，这样就消除了为数据类型中成员容器提供直观名字的需求。就像在本书中所看到的，我们通常倾向于用字符串“COL”给数据类型成员起别名，于是，每一个成员都构成了参照这些数据类型的表的列的基础。在CODE数据类型中命名数据类型成员为CODE会导致对它的引用看似多余（如CODE.CODE）。通过命名类型成员为COL，使引用变为CODE.COL，这就比较清晰了。可见，重要的是数据类型名字，而不是它的成员的名字。

布尔数据类型应该总是强制型的并提供缺省值，但是 Oracle 8i在类型一级不支持缺省值和检查约束，尽管已计划在以后的版本中实现对这种类型的支持。

在定义布尔数据类型时，我们就已经考虑了这一点。下面(代码 6-2中)将看到，我们已给每一个布尔类型加了一个“BOOL”前缀。每一个后缀标识出支持这种类型的允许值。所提供的第一个值就是该类型列的缺省值。

比如，在“BOOL_YNX”类型上实例化的列需要一个缺省值‘Y’和另两个允许值‘N’、‘X’，Y代表是或真，N代表否或假，‘X’用于可选的布尔型标志。当每一行都包含一个值时，查询效果比较好，因为这些值可以做索引；另外，不用“NULL”而是分配一个实际的值，会免除所有含糊不清的情况。

代码 6-2

```
CREATE OR REPLACE TYPE BOOL_YNX AS OBJECT (
  COL          CHAR (1))
/
```

```
CREATE OR REPLACE TYPE BOOL_NYX AS OBJECT (
  COL          CHAR (9))
/
```

```
CREATE OR REPLACE TYPE BOOL_XYN AS OBJECT (
  COL          CHAR (1))
/
```

```
CREATE OR REPLACE TYPE BOOL_YN AS OBJECT (
```

```
COL          CHAR (1))
/
CREATE OR REPLACE TYPE BOOL_NY AS OBJECT (
COL          CHAR (1))
/
```

布尔域作为数据类型不能完全实现，因为需要缺省值作为说明布尔域的成分。Oracle Designer提供了指定缺省值的功能，而数据库中的类型不能指定缺省值。缺省值是在表中列的实例上实现的。因此，为了在数据库级用缺省值实现一个域，需要两步或多步，第一步定义对象类型，第二步当每列在对象类型上实例化时实现缺省值。此外还可能有第三步，用检查约束来定义有限值集。如果你从 Oracle Designer中生成自己的DDL，这三步都可以自动完成。

当建立缺省值为“是”的布尔域的例子时，我们可用代码 6-3所示的过程。

代码 6-3

```
CREATE OR REPLACE TYPE BOOLEAN_Y AS OBJECT (
COL          CHAR (1))
/

CREATE TABLE PERSON (
EMPNO        NUMBER (10) NOT NULL PRIMARY KEY,
NAME         VARCHAR2 (60) NOT NULL,
MALE_YN      BOOLEAN_YN DEFAULT BOOLEAN_YN ('Y'))
/
```

从这个例子可以看出，实现过程首先是建立 BOOLEAN_Y对象类型。你可能会觉得既然无法将缺省值存储为对象类型的一部分，就应该只创建一个布尔对象类型，它应用于所有场合。但由于 Oracle Designer坚持缺省值作为域的一部分，而且它还可以生成合适的 DDL来满足数据结构和缺省值的需要，所以用一对一方式创建同 Oracle Designer域相对应的对象类型是有道理的。

注意 可以基于列类型指定列的缺省值，但在诸如 V.8.0.4的DML语句中不支持缺省值。

3. VARCHAR2

VARCHAR2域由支持各种长度信息的变长字段组成。根据下表，尽量限制不同字段的数目：

VARCHAR2(10)	用于短的名字
VARCHAR2(20)	用于屏幕显示的名字，因为 40个字符的名字占用了报表和屏幕的大部分空间
VARCHAR2(40)	用于现实世界对象的名字
VARCHAR2(2000)	用于描述

对短的描述，过去常用 20、500、1000这些较小的域，但实际上我们发现这样做维护起来很不方便，而且在屏幕和报表中也没有效果。一个存储 80个字符的 VARCHAR2(2000)字段所需的空間和一个存储80个字符的 VARCHAR 2 (250) 字段所需的空間是一样的。

我们使用一个单独的电话号码域 (VARCHAR2(40))，它拥有一个复杂的格式触发器以强制正确的输入。通常，用户只是对查找电话号码感兴趣。如果电话号码存放在多个字段中，用户常常产生很多数据输入错误，这样在多个字段中存储电话号码的优点就都丧失了。

对大多数系统来说，我们不把地区代码和电话号码分开存储，因为用户需求并不要求这么做。当然，如果工程是为电话公司设计的，就要将电话号码的各个组成部分存储在不同的字段中。

电子邮件和Web地址（每个地址250个字符，且没有格式掩码）分配到一个域。用较大的长度是为了保证该域能满足Web地址可能的最大长度。

人名的处理比表面上复杂一些，需考虑姓、名、称呼、职务、职称、绰号等的顺序以及人只有一个名字的可能性。一般的方法是尽量将上面提到的所有字段都包括到系统中，这包括了称谓（Mr.、Mrs.、Ms.、Dr.）姓、名、职称（Ph.D.、MD、CPA等）以及绰号（用于非正式场合）。对人事系统来说，所有这些要集中在一起构成合适的结构。

问题是这种方法不能用于正式的商业往来和顾客名录，因为收集这些信息的人通常不会很仔细地收集所有的信息并正确地登记。在非人力资源系统中使用这种详细的名字结构会带来潜在的问题，使得同一个人在数据库中多次出现。经常是当名字查不到时就再输入一遍。因此，我们倾向于在客户和顾客联系数据库中使用简化的结构，即只使用姓、名、称呼、以及职称，通常这就够了。

实施强制型VARCHAR2字段很困难，特别是对于描述字段。用户并不总是输入描述，如果描述是强制型的，那么用户需要输入一个空格。如果编写了一段验证触发器代码，使得单个空格是非法值，那么可以使用句号来代替。如果不允许使用句号，就使用“X”。如果强制字段长度为10~15个字符，就要使用一个字符串。通过强制型约束来强制描述字段是徒劳的，这样会使数据库被非法数据搞乱。我们建议不要实施强制型的 VARCHAR2数据。

根据特定的系统需求，可能需要一个 VARCHAR2域，其格式限制为数字。这对产品号、合同号、车辆ID号等很有用。注意，一些“号码”可包括嵌入的字符如“20Ta”。在系统开始建设以后，重新把数据类型由数字修改为字符的代价是很大的。因此如果输入的数据不是用于计算，那么这些数据就应该用CHAR或VARCHAR2字段来保存。

推荐的VARCHAR2数据类型见表6-4和代码6-4。

表6-4 推荐的VARCHAR2数据类型

名 字	类 型	长度	精确度	Min	Max	值	缺省值
VC2_10	VARCHAR2	10					
VC2_20	VARCHAR2	20					
VC2_40	VARCHAR2	40					
VC2_2000	VARCHAR2	2000					
Phone	VARCHAR2	40					
Email Web	VARCHAR2	250					

代码 6-4

```
CREATE OR REPLACE TYPE VC2_10 AS OBJECT (  
COL          VARCHAR2 (10))  
/
```

```
CREATE OR REPLACE TYPE VC2_20 AS OBJECT (  
COL          VARCHAR2 (20))  
/
```

```
CREATE OR REPLACE TYPE VC2_40 AS OBJECT (  
COL          VARCHAR2 (40))  
/  
  
CREATE OR REPLACE TYPE VC2_2000 AS OBJECT (  
COL          VARCHAR2 (2000))  
/
```

有时，我们需要定义有特殊意义和用途的数据类型，比如，我们可以使用 VC2_40域作为电话号码，但为了使数据模型和多用途的文档更清晰，我们增加另一个称为 PHONE_TX的数据类型，详见代码6-5。

代码 6-5

```
CREATE OR REPLACE TYPE PHONE_TX AS OBJECT (  
COL          CHAR (40))  
/  
  
CREATE OR REPLACE TYPE WEB_TX AS OBJECT (  
COL          CHAR (250))  
/
```

虽然把这两种数据类型分开对目前的应用程序没有带来特别的好处，但这样做却使得我们实现的系统能够平滑地实现将来的升级，如增加检查约束和附加于该数据类型的缺省值。这就是区分数据类型VC2_40和PHONE_TX的关键。我们可在检查约束内写一段代码，当输入电话号码时，这段代码充当了格式掩码，验证电话号码串的结构。这个检查约束只作用于PHONE_TX类型的列，而不作用于VC2_40类型的列。

因此，当能够把检查约束加到数据类型上时，要做的唯一的改动就是把检查约束加到PHONE_TX类型上。因PHONE_TX类型已分配给整个数据库相应的列了，新的检查约束会在整个系统范围内自动继承。

6.2.2 日期

在Oracle中，日期以数值字段来存储，精确到秒。一天的长度正好为 1.000，分和秒是天中的小数部分。在存储日期时如果对事件发生的具体时间不感兴趣，可以截断输入的日期信息，只存入其中的整数部分。这能保证存入的日期是整数，且代表事务发生日的午夜。

可以有两个域：一个用于日期，另一个用于日期及具体时间。把日期和时间分开存储是没有道理的。如果需要在应用中分别显示日期和时间，可以将同一字段显示两次，每次使用不同的格式掩码。

因采用数值存储方式，在Oracle数据库中，无需考虑2000年问题。Oracle 采用这种方式存储时间信息已有很多年了。只要日期存储在“Date”类型的字段中，就没有2000年问题需要解决。

如果存储的日期信息不是精确到秒，就会有一些问题。比如，当在一个字段中出现 1月7日午夜12点这个日期值时，无法判断它所指的是 1月7日这一天还是1月7日午夜。不附带具体时间不能指定日期。虽然在应用程序中可以不显示具体时间，但它却存在于数据库中。当日期精度水准定位在月或季度时，如杂志的出版日期只列出年、月，这样处理会带来问题。存

储什么来表示1999年7月？如果我们知道日期精确到“季度”，这没有什么问题。但就某天出版的周刊而言，怎么处理呢？可以创建一个单独的字段来存储周期信息，或者为每个不同类型的周期创建一个字段。我们喜欢在一个单独的字段里存储周期信息，这样如果要进行修改，也无需改动数据模型。当然，也可以选择建立单独的域分别用于以月为周期的日期和以季度为周期的日期。

另一个与日期有关的情况是对日期信息的范围限制。日期域上的范围检查应根据系统日期，这样范围就不会过时。例如，如果正在使用员工的出生日期，你可能想使用 SYSDATE-100年。如果维护公司员工的历史信息有很长一段时间了，这样做也可能有问题。不要假定系统40年后就不用了。过去谁会想到60年代建设的系统在2000年还在使用？

你也许想根据特定的开始日期创建日期范围，如对于出生日期而言，你可能想使用系统中所存人员的最早出生日期；对事务日期而言，范围应从系统中目前最早的事务日期开始，跨越整个系统日期，或者是系统日期加上登录新事务的某个时间段。例如，在一个涉及工程规划的带有里程碑概念的系统，用于里程碑的恰当的域以遗留系统传递过来的信息的最早日期作为起点，到系统日期加上20年为止，因为客户正在准备的计划跨越很大的时间范围。

年代（财政年或税务年）的存储通常不使用日期字段。年代使用4个字符字段来存储。这样最大的年代为公元9999年。

对于日期的存储，我们建议使用表6-5所示的域。

对日期和时间，特别要强调是应该尽可能地使用范围验证，确保错误的事务日期不能输入系统（如02年）。

第一个日期域需要数据库触发器来实现日期值的截断。如果一个应用程序使用 SYSDATE 作为缺省值，插入的值会存储事务发生的时间，如果这个信息不恰当，可以通过一个在该列上执行TRUNC操作的数据库触发器来删除它。

表6-5 建议的日期域

名 字	类 型	长度	精确度	Min	Max	值	缺省值
Date	Date					Truncate to Midnight	
Month	Date					Truncate to Lowest Month	
Qtr	Date					Truncate to lowest quarter	
Time	Date						
Year	Number	4		1900	Sysdate Year+30		
Birth Date	Date			1900	sysdate		
Emp Birth date	Date			1900	Sysdate- 15 years		
Tran date	Date			1970 or earliest transaction date	Sysdate+ max lead time		Sysdate
Plan date	Date			1970	Sysdate+ 30 years		

域对象类型如代码 6-6所示。

代码 6-6

```
CREATE OR REPLACE TYPE DATE_MIDNIGHT AS OBJECT (  
COL          DATE)  
/
```

注意，从数据库的角度来看，定义这个域对象类型还没有带来任何好处，我们本来可以将该列的数据类型简单地定义为 DATE 型。但是，同本章前些部分谈到的布尔的实现一样，通过这种方法，相应的 Oracle Designer 域会比数据库域更灵活。域名指示出这个域有附加的验证，尽管从物理上看验证是加在列上，而不是加在该对象类型的成员上的。

下面代码 6-7 使用 DATE_MIDNIGHT 对象类型和一个数据库触发器来确保事务日期仅记录事务的月、日和年。

注意 数据库触发器不支持 Object 或 Collection 类型：对于 Oracle 8i，NEW 列要修改。

下面这个例子的语法是正确的，但是目前还不被支持。

代码 6-7

```
--***Example_06_06 is prerequisite  
CREATE TABLE TRANS (  
TRANS_ID      NUMBER (10) NOT NULL PRIMARY KEY,  
TRANS_DATE    DATE_MIDNIGHT NOT NULL)  
/  
  
CREATE OR REPLACE TRIGGER BIU_TRANS  
BEFORE INSERT OR UPDATE  
ON TRANS  
FOR EACH ROW  
BEGIN  
:NEW.trans_date.col := trunc(:new.trans_date.col);  
END;  
/
```

要注意的是 Oracle Designer 不会为你生成上面的这个触发器。你很可能将这条规则作为 Oracle Designer 域描述的一部分存储起来，然后在物理实现时建立这个触发器。

DATE_MONTH 和 DATE_QUARTER 数据类型可以采用与 DATE_MIDNIGHT 数据类型同样的方法来创建，使用的语法如代码 6-8、6-9 所示。注意基于日期的列存放的值采用的是完整的日期格式，不管你是使用其一部分还是使用其所有信息。例如，当执行下面的 TRUNC 操作时，DATE_QUARTER 域返回一个值 '01-JUL-1998'。不幸的是，我们必须编写代码(如下所示)，而不能把缺省登录同域相连以便自动执行 TRUNC。

```
TRUNC('18-AUG-1998','Q')
```

该代码不会返回一个表示季度的数值型的值（即 '1' 表示 1~3 月份时间段，'2' 表示 4~6 月份时间段等）。参见代码 6-8 和 6-9。

1. 时间

时间域同日期数据类型一样，也没有限制(如代码 6-10 所示)。由于缺省值和附加的数据检查，你将创建与 Oracle Designer 中的域直接对应的对象类型。此时，可能希望创建两个域，一个域

代码 6-8

```
CREATE OR REPLACE TYPE DATE_MONTH AS OBJECT (  
COL          DATE)  
/  
CREATE TABLE TRANS (  
TRANS_ID     NUMBER (10) NOT NULL PRIMARY KEY,  
TRANS_DATE   DATE_MONTH NOT NULL)  
/  
  
CREATE OR REPLACE TRIGGER BIU_TRANS  
BEFORE INSERT OR UPDATE  
ON TRANS  
FOR EACH ROW  
BEGIN  
:NEW.TRANS_DATE.COL := TRUNC(:NEW.TRANS_DATE.COL, 'MON');  
END;  
/
```

代码 6-9

```
CREATE OR REPLACE TYPE DATE_QUARTER AS OBJECT (  
COL          DATE)  
/  
CREATE TABLE TRANS (  
TRANS_ID     NUMBER (10) NOT NULL PRIMARY KEY,  
TRANS_DATE   DATE_QUARTER NOT NULL)  
/  
  
CREATE OR REPLACE TRIGGER BIU_TRANS  
BEFORE INSERT OR UPDATE  
ON TRANS  
FOR EACH ROW  
BEGIN  
:NEW.TRANS_DATE.COL := TRUNC(:NEW.TRANS_DATE.COL, 'Q');  
END;  
/
```

个域的缺省值为 SYSDATE，另一个域没有缺省值。这两个域除了名字以外其他方面均相同，尽管引用它们的列可能含有基于域的 Oracle Designer 版本的缺省值。

代码 6-10

```
CREATE OR REPLACE TYPE DATE_TIME AS OBJECT (  
COL          DATE)  
/  
  
CREATE OR REPLACE TYPE DATE_TIME_SYSDATE AS OBJECT (  
COL          DATE)  
/  
  
CREATE TABLE PERSON (  
EMP_ID       NUMBER (10) NOT NULL PRIMARY KEY,  
ENAME        VARCHAR2 NOT NULL,
```

```
HIRE_DATE      DATE_TIME,
CREATED_DATE   DATE_TIME_SYSDATE NOT NULL,
MODIFIED_DATE  DATE_TIME SYSDATE NOT NULL)
/
```

2. 年代

年代域可用 NUMBER(4)来实现(如代码 6-11所示)。可以用一个检查约束来保证值是在 1900~2100之间。采用这种方法,可以确保输入的年代是在可接受的范围之内。

代码 6-11

```
CREATE OR REPLACE TYPE YEAR AS OBJECT (COL          number (4),
MAP MEMBER FUNCTION YEAR_MAP RETURN NUMBER)
/

CREATE OR REPLACE TYPE BODY YEAR IS
MAP MEMBER FUNCTION YEAR_MAP RETURN NUMBER IS
BEGIN
RETURN COL;
END;
END;
/

CREATE TABLE SEMESTER (
CLASS_YEAR          YEAR NOT NULL CHECK (CLASS_YEAR.COL BETWEEN 1900
AND 2100),
SEMESTER_CD         VARCHAR2 (5))
/

CREATE OR REPLACE TRIGGER BIU_SEMESTER
BEFORE INSERT OR UPDATE
ON SEMESTER
FOR EACH ROW
DECLARE
INVALID_YEAR      EXCEPTION;
BEGIN
IF :NEW.CLASS_YEAR.COL BETWEEN 1900 AND 2100
THEN NULL;
ELSE RAISE INVALID_YEAR;
END IF;
EXCEPTION
WHEN
INVALID_YEAR THEN
raise_application_error('-20001','CLASS_YEAR IS NOT A VALID YEAR
BETWEEN 1900 AND 2100.');
```

3. 出生日期

对于出生日期 (DOB) 域,只需要创建这个域,然后在该域实例化的每一列上创建一个检查约束。以下代码 6-12表示出生日期域和该域的一次实例化,包括一个检查约束。

要注意的是检查约束从 SYSDATE中减去了 5475天,即15年,因为我们确信没有一个员工的年龄会小于16岁。

代码 6-12

```
CREATE OR REPLACE TYPE DOB AS OBJECT (  
COL          DATE)  
/  
  
CREATE TABLE EMP (  
EMP_ID      NUMBER (10) NOT NULL PRIMARY KEY,  
NAME_TX     VARCHAR2 (60) NOT NULL,  
BIRTH_DATE  DOB CHECK (BIRTH_DATE.COL > SYSDATE-5475,  
              'DD-MON-YYYY'))  
/
```

4. 事务日期

事务日期域需要一个对象类型和一个作用于域的每一次实例化的 BEFORE INSERT 数据库触发器(如代码 6-13 所示)。因为这个检查需要对最早记录的事务日期的验证, 所以它不能在检查约束内执行。这意味着我们必须对表进行一个查询, 来决定事务范围的下界。只创建一个 BEFORE INSERT 触发器的原因在于事务日期不允许更新。

代码 6-13

```
CREATE OR REPLACE TYPE TRAN_DATE AS OBJECT (  
COL          DATE,  
MAP MEMBER FUNCTION TRAN_DATE_MAP RETURN DATE)  
/  
  
CREATE OR REPLACE TYPE BODY TRAN_DATE IS  
MAP MEMBER FUNCTION TRAN_DATE_MAP RETURN DATE IS  
BEGIN  
RETURN COL;  
END;  
END;  
/  
  
CREATE TABLE TRANS (  
TRANS_ID     NUMBER (10) NOT NULL PRIMARY KEY,  
TRANS_DATE   TRAN_DATE NOT NULL)  
/  
  
CREATE OR REPLACE TRIGGER BI_TRANS  
BEFORE INSERT  
ON TRANS  
FOR EACH ROW  
DECLARE  
CURSOR C1 IS  
SELECT MIN(TRANS_DATE) FIRST_TRAN  
FROM TRANS;  
INVALID_DATE EXCEPTION;  
V_DATE   DATE;  
BEGIN  
FOR C1_REC IN C1 LOOP  
V_DATE := C1_REC.FIRST_TRAN.COL;
```

```

IF :NEW.TRANS_DATE.COL BETWEEN V_DATE AND SYSDATE + 3650 T
HEN
NULL;
ELSE RAISE INVALID_DATE;
END IF;
END LOOP;
EXCEPTION
WHEN INVALID_DATE THEN
raise_application_error('-20001', 'TRANSACTION DATE MUST BE
BETWEEN '
||TO_CHAR(V_DATE)||' AND
||TO_CHAR(SYSDATE + 3650)
||',');
END;
/

```

这个触发器确保新输入的事务日期大于所记录的第一个事务的日期，小于当前日期后推三年所得的日期。

5. 财政事务域

代码6-14所示的财政事务域同前面定义的事务域几乎完全一样，唯一的区别是它所允许的将来事务范围可以达到未来30年或更远。

代码 6-14

```

DROP TABLE FIN_TRANS;
DROP TYPE FIN_TRAN_DATE;
CREATE OR REPLACE TYPE FIN_TRAN_DATE AS OBJECT (
COL
DATE,
MAP MEMBER FUNCTION DT_MAP RETURN DATE)
/

CREATE OR REPLACE TYPE BODY FIN_TRAN_DATE IS
MAP MEMBER FUNCTION DT_MAP RETURN DATE IS
BEGIN
RETURN COL;
END;
END;
/

CREATE TABLE FIN_TRANS (
TRANS_ID          NUMBER (10) NOT NULL PRIMARY KEY,
TRANS_DATE        FIN_TRAN_DATE NOT NULL)
/

CREATE OR REPLACE TRIGGER BI_TRANS
BEFORE INSERT
ON FIN_TRANS
FOR EACH ROW
DECLARE
CURSOR C1 IS
SELECT MIN(TRANS_DATE) FIRST_TRAN
FROM FIN_TRANS;

```



```
INVALID_DATE EXCEPTION;  
V_DATE DATE;  
BEGIN  
FOR C1_REC IN C1 LOOP  
V_DATE := C1_REC.FIRST_TRAN;  
IF :NEW.TRANS_DATE BETWEEN FIRST_TRAN AND SYSDATE + 10950 THEN  
NULL;  
ELSE RAISE INVALID_DATE;  
END IF;  
END LOOP;  
EXCEPTION  
WHEN INVALID_DATE THEN  
RAISE_APPLICATION_ERROR('-20001','TRANSACTION DATE MUST BE  
BETWEEN' || TO_CHAR(V_DATE) ||  
' AND ' || TO_CHAR(SYSDATE + 10950) || '.');  
END;  
/
```

6.2.3 数值

数值是最难确定的域。当使用数值时，有很多不同的验证级别和精度级别。例如，数值包括如下类型：

计数（整数）。

包括百分比的连续量。

货币。

对于数值字段，我们不仅需要关心最小长度和最大长度，还要注意精度级别，即小数点后面数字的位数，特别是需要很多不同的数值范围和精度域。最初我们认为把域限制为正数是一个好的办法，但事实上，这很少会出现。例如，通常认为库存量为正数，退回的货物应记为负数。一般常把正数的限制加在作为主码的数上，这些主码常选择序号，因此这样就使约束变得毫不相干。一般情况下，不需要给数指定范围，因为范围在字段长度中已经隐含了。

1. 货币

货币的处理是一项最有趣的工作。在美国，用 NUMBER14,2来表示货币类型是很容易的。但用这种表示法也会有一些问题，外国货币可能需要更多或更少的数字，例如，若系统存储的事务使用的单位是日元，则通常不需要小数点后的数字，而意大利里拉或俄罗斯卢布可能会需要更大的最大值。即便是对美国货币，也可能并不需要将数额记录得过于详细，而仅需记录到美元或者仟美元这一级。

另一个复杂情况是数据库中的货币数字在计算时常常需要精确到小数点后两位或更多，如某些金融机构的货币计算。

不能简单地认为货币值不能为负值。因此，应考虑使用负值来支持事务处理以及帐面净收入为负值的情况。

以下的域适合美国货币：

N6_2——NUMBER 6,2。

N14_2——NUMBER 14,2。

所用代码如 6-15 所示。

根据特定的系统，可能需要两个同样的域，这两个域用小数点后一定位数的有效数字来

限制。甚至还可以指定值必须处于比整数大的间隔内，例如，NUMBER (6, -3) 要求值用 1000 的间隔来存储，这对至少需要 1000:1 汇率的国际货币非常有用。这就意味着要支持所有的国际货币，应使用以下的域：

NO10_2——NUMBER 10, 2。

NO20_2——NUMBER 20, 2。

使用代码 6-16 可达到这一目的。

代码 6-15

```
CREATE OR REPLACE TYPE N6_2 AS OBJECT (
COL          NUMBER (6,2))
/
```

```
CREATE OR REPLACE TYPE N14_2 AS OBJECT (
COL          NUMBER (14,2))
/
```

代码 6-16

```
CREATE OR REPLACE TYPE N10_2 AS OBJECT (
COL          NUMBER (10,2))
/
```

```
CREATE OR REPLACE TYPE N20_2 AS OBJECT (
COL          NUMBER (20,2))
/
```

建议用于货币的域要完全根据系统的具体情况而定。一个适合于大型纽约经纪行的域同一个适用于小零售商的域是不同的。

已经使用的货币域的列表见表 6-6。一个系统不太可能会需要用到表中列出的所有域。另外，还可以发现这些数据类型同上述一般的数据类型在结构上有部分重叠。这在现在看起来似乎毫无必要，但它却会无缝地包容以后的特征。根据本章提到的语法，能比较容易地创建这些域。

表 6-6 货币域

名 字	类 型	长度	精确度	Min	Max	值	缺省值
Small amt.	Number	10	2				
Large amt.	Number	14	2				
Small amt. precise	Number	16	6				
Large amt. precise	Number	18	6				
Intl small amt.	Number	12					
Intl large amt.	Number	16					
Intl small amt. precise	Number	16	4				
Intl large amt. precise	Number	20	4				

2. 百分比

百分比和货币有许多相同的复杂性。很容易想到将百分比表示为 2, 0 字段。但是，根据应用的实际情况，可能会需要更高的精度。百分比是可以超过 100% 的，很多应用仅需要 1% 域。

使用5,2或6,2字段表示百分比通常是安全的，除非应用程序是用于金融机构的，金融机构可能会需要10,5或15,10字段表示百分比。

我们使用的百分比域见代码 6-17和表6-7。

代码 6-17

```
CREATE OR REPLACE TYPE SMALL_PCT AS OBJECT (  
COL          NUMBER (5,2))  
/  
  
CREATE OR REPLACE TYPE LARGE_PCT AS OBJECT (  
COL          NUMBER (6,2))  
/  
  
CREATE OR REPLACE TYPE SMALL_FIN_PCT AS OBJECT (  
COL          NUMBER (10,5))  
/  
  
CREATE OR REPLACE TYPE LARGE_FIN_PCT AS OBJECT (  
COL          NUMBER (15,10))  
/
```

表6-7 百分比域

名 字	类 型	长度	精确度	Min	Max	值	缺省值
Pcnt	Number	3		-100	100		
Big pcnt	Number	5					
Pcnt precise	Number	5	2	-100	100		
Big Pcnt	Number	7	2				
Precise							
Finance Pcnt	Number	15	5				

3. 其他数值

我们需要几个域来支持总体意义上的数值。为避免混淆，百分比和货币域不要用于非货币或非百分比的数值，以下的域用于数值：

```
SMALL_INT      Number 5  
LARGE_INT      Number 10  
SMALL_NR       Number 10,5  
LARGE_NR       Number 10
```

科学型的应用程序可能需要更高的精度，且会需要另外的域。代码 6-18列出了这些域的代码。

代码 6-18

```
CREATE OR REPLACE TYPE SMALL_INT AS OBJECT (  
COL          NUMBER (5))  
/  
  
CREATE OR REPLACE TYPE LARGE_INT AS OBJECT (  
COL          NUMBER (10))  
/
```

```
/

CREATE OR REPLACE TYPE SMALL_NR AS OBJECT (
COL          NUMBER (10,5))
/

CREATE OR REPLACE TYPE LARGE_NR AS OBJECT (
COL          NUMBER (10))
/
```

6.3 其他数据类型

这里没有提到的数据类型不足以用域来实现。Long text、OLE text、pictures、movies和files通常不用域来支持。

6.4 结论

我们在本章中列出的域应支持绝大多数的业务系统，不能不加思索地应用域。你可能希望使用比上面所说的更多的域，也可能只使用其中的一部分。对于任何特定的系统，可能需要更多的域来支持较高的精度级别（如科学计算），但是我们发现对绝大多数的业务系统来说这些列表已经足够了。