

第5章 命名约定

对任何系统来说，应用的一致性和恰当的命名约定是成功的关键因素。必须采用标准来实现一致性。在建立命名标准时，应牢记以下的一般约定：

1) 选取的名字必须清晰，并且对使用者来说是容易理解的。比如，就数据仓库而言，最终用户是使用者；但是对一个物理数据库而言，使用者包括系统专业人员和开发人员。名字对所有这些人员都合适可能非常困难。

2) 名字不应太长。选取的名字需要多次输入，长的名字不仅增加了输入的时间，也增加了出错的概率，甚至在处理已生成的代码时，长的名字也可能使得代码难以理解。有些产品允许选取变长的名字，但却只能显示前 10 个左右的字符。比如，在 Oracle Designer 的某些部分中，当给一个属性分配一个域名时，长的域名会被截短。

3) 名字应尽可能地表示出对象的有意义的信息。通常给系统元素的名字加上前缀和后缀，这样对名字的分类组织非常有用。

注意 在应用约定 2) 和 3) 时，必须进行权衡。为了表示出更多的信息，比较容易的方法就是建立长的名字，但这些名字不易重复使用；另一方面，短的名字仅对最初设计系统的人员来说容易理解，且其本身也比较含糊。

4) 名字应容易记忆。比如，在整个系统中只使用单词的一种形式，不使用单词的词性变化形式（如“-ing”形式、“-ed”形式以及名词的复数形式等）。只使用单词的原形也会使名字变得短些。

5) 数据库任何地方出现的文本信息都需要精确的命名约定，这包括数据本身。比如，在数据库的值列表类中，对所有的主码都采用大写形式。

6 必须对所有的元素同时创建完整的命名标准。不能先创建数据库命名标准接着又创建开发标准，因为这些标准可能会相互制约。标准必须是为整个开发环境而设立的。本书因描述的是数据库的设计，所以只提到了有关数据库的决定。但是，在传统的开发环境中，框架对象(弹出式列表、复选框等)、应用文件名等都需要命名约定。

在数据库中，有很多元素都需要一致的命名约定。把这些元素分为两类。

1) 在逻辑（数据模型）方面，类图中出现的元素都需要命名：

模式。

类图。

类/表/实体。

属性。

域。

操作/方法。

2) 在物理实现方面，下列元素需要命名：

序列。

短表命名别名。

约束。

索引。

对象类型。

表别名。

下面将探讨对这些系统元素所建议的命名约定。

5.1 逻辑 (数据模型) 命名约定

系统可能非常大，可包括成百个表，最大的系统甚至有 1000个表。正是在这些大系统中，经常会看到诸如 AX145_6这样令人讨厌的表名。需要把使用的系统划分为一些可管理的部分和模式。

1. 模式的命名

模式是表的集合，这些表既可以作为数据库也可以作为独立的用户帐户来实现。模式允许我们在整个系统中对表进行引用。在数据模型中，对每一个主体域都应有一个模式。模式的数量不应太多。一个模式应包括大约 30~50个表。为了便于管理，单个模式包含的表不应多于100个。

模式应采用很短的缩写来命名。因为大部分系统的模式都不足 10个，因此两个字母的缩写就足够了，比如用“CO”表示“Core”，用“FI”表示“Finance”。应尽可能地使用模式逻辑名字的头两个字母。因为要使用“模式名.表名”这种约定，所以模式名应保持很短。

在多对多联系中，关系中的每一个表都位于不同的模式中，当表由这些多对多的联系导出时，也会带来一些复杂性，问题就在于把这些有交叉的表置于何处。一个可能实现的算法就是用最逻辑的表来保存关联表。例如，如果正在给一个工程分配人员，那么就用包含 Project表的模式来保存关联；但是，如果正在给人员分配设备，那么就用包含 People表的模式来保存关联。

雇员 (Employee) 和部门 (Department) 之间的雇佣关系中也存在同样的问题，这时的关联应该存储在那里呢？答案不是很明确。用雇员表或部门表来保存雇佣关联表都是有效的，所以，我们使用不同的算法来保存关联表。当存在一个关联表且形成关系的两个表存在于两个不同的模式中时，关联表应该存放在其表名按字母顺序靠前的表中，例如，在 DEPT/EMP 关系中，如果 DEPT表和EMP表存在于两个不同的模式中，则应该把雇佣关系保存在包含 DEPT表的模式中，因为在字母顺序上 DEPT比EMP靠前。这是一个比较随意的规则，但是还没有一个逻辑规则来一致、准确地决定关联表隶属哪个模式。

SQL有一个命令CREATE SCHEMA，该命令用来把多个DDL语句组合成一条单独的语句。没有单独的DDL语句会被数据库执行，除非 CREATE SCHEMA语句的所有组成部分都有效。本质上，“模式”是指存在于单个Oracle数据库帐户中的DDL语句的集合。不能创建独立于数据库帐户的模式。事实上，CREATE SCHEMA语句的语法引用了拥有对象的数据库帐户的名字。CREATE SCHEMA语句没有自己的名字，因为它只在逻辑意义是单独的集合。下面的代码5-1用来为数据库帐户SCOTT建立DEPT表。

CREATE SCHEMA语句对一个特定的数据库帐户可执行若干次。可以写出以下代码 5-2所示的创建EMP表的CREATE SCHEMA语句，该表含有一个参照DEPT表的外码。

代码 5-1

```
CREATE SCHEMA AUTHORIZATION SCOTT
CREATE TABLE DEPT (
DEPT_ID          NUMBER(2)      NOT NULL PRIMARY KEY,
NAME_TX          VARCHAR2(14) NOT NULL,
LOC_TX           VARCHAR2(13) NOT NULL)
/
```

代码 5-2

```
CREATE SCHEMA AUTHORIZATION SCOTT
CREATE TABLE EMP (
EMP_ID           NUMBER(4) NOT NULL PRIMARY KEY,
NAME_TX          VARCHAR2(10) NOT NULL,
JOB_TX           VARCHAR2(9),
EMP_ID_RFK       NUMBER(4),
HIRE_DT          DATE NOT NULL,
SAL_NBR          NUMBER(7,2) NOT NULL,
COMM_NBR         NUMBER(7,2),
DEPT_ID          NUMBER(2) NOT NULL REFERENCES DEPT (DEPT_ID))
/
```

但是，当试图在一条CREATE SCHEMA语句中创建多个表时，CREATE SCHEMA语句会产生下面的错误，如代码5-3所示。

代码 5-3

```
CREATE SCHEMA AUTHORIZATION SCOTT
CREATE TABLE DEPT (
DEPT_ID          NUMBER(2)      NOT NULL PRIMARY KEY,
NAME_TX          VARCHAR2(14) NOT NULL,
LOC_TX           VARCHAR2(13) NOT NULL)

CREATE TABLE EMP (
EMP_ID           NUMBER(4) NOT NULL PRIMARY KEY,
NAME_TX          VARCHAR2(10) NOT NULL,
JOB_TX           VARCHAR2(9),
EMP_ID_RFK       NUMBER(4),
HIRE_DATE        DATE NOT NULL,
SAL_NBR          NUMBER(7,2) NOT NULL,
COMM_NBR         NUMBER(7,2),
DEPT_ID          NUMBER(2) NOT NULL REFERENCES DEPT (DEPT_ID))
/
```

ERROR at line 9:
ORA-00942: table or view does not exist

问题出在试图创建一个EMP表时，表EMP同时含有一个参照DEPT表的外码。表DEPT必须物理上实例化，才允许其他表（例如EMP表）用一个外码来参照它。CREATE SCHEMA语句在执行过程中不能实现完整性约束的验证。

2. 类图/ERD的命名

很多类图可用来描述系统。比如，一个包含了整个系统的图可称为“企业（Enterprise）”。

每个模式可包含一个类图。然而，由于模式可多达 100 个类，因此每个模式可能需要不止一个类图。如果需要比较小的图，就用模式名加上一个描述性图名。可以假设我们处于一个类似于 Windows 的环境中，在名字的选择方面可以不受限制，比如，想要在 CORE 模型中显示同 Security 关联的表，表名就可以是 CO1 Security (“CO”指的是核心模式，“1”代表模式的优先号，“Security”描述包含在图中的表)。

图名应该是有意义的，同样的类可在多个类图中出现。当建立子图时，主体域应包括所有相关类。显示一个关联类的任何类图应显示初始的多对多关联的双方。在类图中被任何类参照的值列表类（参照表）都应表示出来，比如，在 Project 类的类图中，Project Type 类和 Project Status 类都应表示出来。

版本。当通过 SDLC 进行时，图可能会有很多不同的版本。版本信息应包含在图的名字中。我们推荐给图的名字加上后缀，该图的名字包括系统正在运行的版本、工程的生命周期阶段以及图的版本。在工程阶段代码中，“STR”表示策略、“ANA”表示分析、“DEV”表示开发、“PRO”表示生产。以下是完整的类图名字的例子：

ENTERPRISE V1 ANA1。表示在系统第 1 版的分析阶段的企业数据模型的第 1 版。

CO ACCESS CONTROL V2 DEV3。表示数据模型的核心部分的第 3 版，用来在系统第 2 版的开发阶段处理访问控制子系统。

这些图名似乎并不精巧，但是它们是不用输入的，只需从列表中选择即可。因为只是文件名，所以这些名字只在系统中出现一次。

3. 类/表/实体的命名

好的类名是很重要的，因为这些名字常常直接翻译成表的名字，目的就是要实现类名同表名的匹配。在 Oracle 界中，有一个比较显著的令人遗憾的约定，即用名词的单数形式来命名实体而用复数形式来命名表。这样，在查看一个 ERD 时，表名和实体显示的名字是不一样的。从而在分析时用的是一个名字，但相关的数据库表使用的却是另外一个名字。同样地，在 ERD 中可以采用比较长的名字，但在数据库的执行过程中，又采用了长词的缩写。其实这种区分倒比较有用，因为长的名字更容易向用户显示，但对开发者来说用处却不大。模型名和表名之间出现的这种不匹配的情形会引起很多混淆。

我们推荐从一开始就保持类和表拥有同样的名字。为了向用户显示比较有意义的名字，缩写单词要能转换成长词的形式。在 Oracle Designer 中有一个工具，它能自动地进行缩写词到长词的转换。

类的名字不像类的描述那样重要，选择类名时应该认识到目前企业文化会影响到名字的选择，而选取的名字对外界来说可能没什么意义。例如，和我们一起合作的一个厂商要把设备故障称为“票 (ticket)”，因为所有出现故障的设备都用一个被称为“票”的文档来记录。在没有遵循我们更好建议的情况下，这个类就被命名为“票”。虽然这种命名对某些用户来说是比较贴切的，但容易引起将来从事系统工程的人员的迷惑。还有一个例子，HMO (家庭医疗组织) 中的“医疗声明 (Medical Claim)”是一个用来表示对医疗服务的付费请求的特定文档。还会有其他的没有声明的服务付费请求，这些请求都已包括在同样的类中了。现在的问题是命名这个类为“声明 (Claim)”还是“服务付费请求 (Payment Request for Service)”。该类最终被称为“声明”，即使它里面还包含其他信息。这两种情形都强调说明了选择清晰的、不会引起混淆的类名的必要性。

名字的选择应符合数据库界普遍考虑的命名方法，而不是其他任何组织或工业的规格标准。类的命名应尽可能地选取最能贴切地描述类所存储的信息的一两个词。第4章提到过的短语同样有助于给类命名，特别是“Each object in this class is a...（该类中的每个对象是一个...）”这类短语，其中省略号处应填类的名字。

1) 缩写。当给系统中的类和其他项命名时，为整个系统设计一个固定的缩略词和单词的对照表是很重要的。在数据库中，每一个名字、类、属性、约束等都使用同样的名字和缩略词对照表。在一些组织内部，不同的部门在不同的环境下使用不同的缩写，因为用户使用不同的名字参照同一个对象。在数据模型中，系统设计人员错误地复制这些不一致性，从而导致混淆。

当缩略词表完成时，它不仅应该包括单词及其缩写，而且还包括单词使用环境的简要描述，这有助于在整个系统中保证一致性。表5-1中列出了一些好的类名、它们的缩写以及描述。注意，所有的类名都用的是单数形式。

决定有效的缩写策略就是找出单词的缩写形式，这种形式能容易地回复成单词的原形。缩略词应展示给几个人看看，试验它是否能回复为同一个单词。同时还应控制缩略词的字符数。大部分情况下都使用五个字母的缩写，这是容易回复单词的最少字母数。有一些标准缩写采用不同的字母数目（如：ST代表街道、ADDR代表地址），在这种情况下，就应使用人们普遍接受的缩写形式。五个字母的缩写是最少输入和清晰意义之间的合理权衡，尽管这样可能会使新的开发人员花费一些时间去适应。

表5-1 类名、缩写及其描述

类 名	缩 写	描 述
EMPLOYEE	EMP	在组织内部全职或兼职工作的人员
DEPARTMENT	DEPT	组织内部雇员的逻辑小组，如财务、销售等
PURCHASE	PO	订购行为
TASK	TASK	由雇员完成的工作流事件
PERSON	PERSON	组织内人员的姓名
ADDRESS	ADDR	组织或人员的地址
MODIFY	MOD	修改数据库记录的行为
MERCHANDISE	MERCH	组织销售的项目
MOVEMENT	MOVE	把商品从一个地点转移到另一地点的行为
SALE	SALE	销售商品换取货币的行为
TRAVEL	TRAVL	雇员从一个地点到另一地点的迁移行为

2) 类的类型。类的类型有很多种，每一种都有其命名约定。

主类。这些类包括主事务以及系统内部所关注的对象，如雇员、销售和部门等。主类用一个或多个有意义的、大约5个字母的、用下划线来连接的单词来命名。如果单词太长，应采用缩写表中的缩写形式，只有缩写表中的词才能使用。如果要使用的词不在目前的缩写表中，就要用缩写表的修改机制将新词添加到缩写表中。在第19章中可以看到这类名字的例子。

值列表类。值列表类和主类大不一样，值列表类只是属性的有效值列表。因为值列表类在概念上不同于其他类，所以把值列表类组织在一起非常有用。值列表类同其他类相比不太常见，再者，因类通常以字母顺序排列，因而最好在属性名前加上一个前缀

Z_，这样，所有的值列表类就自动地归入按字母顺序排列的表的尾部，使其比较容易地得到标识。例如，就一个工程的可能状态的列表而言，值列表类的名字可以是 Z_PROJ_STS；对单词“type”通常可以省略，因为前缀Z_已经标识出该类是一个值列表类。采用这个系统，Z_PROJ表示工程类型、Z_EMP表示雇员类型。当这些类显示给用户时，若加上单词“type”，则对提高清晰度非常有用。通过使用缩略词对照表，对以Z_开头的表用空串来代替单词“type”，我们可以达到这一目的。

冗余类。任何类如果是非规范化导致的聚集类，就必须用X_作为前缀。比如，Ledger Account History（财务流水帐）类是在Ledger帐户中存储每日的事务概要，它被称为X_LEDG_ACCT_HIST。冗余类是删除后可以重构的类。

关联类（交叉实体）。源于多对多关系的类会带来很多难以解决的问题。首先要认识到关联类通常表示一个实物，这个实物可直接进行明确地说明。下面是一些例子：

课程和学生之间的关联可称为“Enrollment（登记）”。

人和设备之间的关联可称为“Assigned Equipment（分配的设备）”。

人员和部门之间的关联可称为“Employment（雇佣）”。

人和角色之间的关联可称为“Assigned Role（分配的角色）”。当关联的一方不能和现实世界的某个事物明显对应时，“Assigned（已分配）”就是一个比较有用的清晰的名称。

几乎在所有情况下，都可以采用连接两个基本类的方法来对关联命名。

明细类。在数据模型中，经常存在明确的主-明细关系，PO-PO Detail和Sale-Sale Detail就是这种关系的两个例子。当一个类实质上是另一个类的明细时，明细类的命名约定是<父类名>_DTL。在上述两个例子中，名字分别是PO_DTL和SALE_DTL。虽有人不愿用_DTL，而愿意用_LINE，但我们还是建议使用后缀_DTL，因为_DTL在某种程度上比较通用。假如正在描述一个定单或销售文档，“line”和“detail”意义是相同的。除了文档以外，还可能存在同样类型的关系。比如，一个任务可完全分解为一个子任务层，这个子任务层称为Task Details，但是可能没有任务文档，这样TASK_LINE就没有意义。

概化类。概化类是其他类的概化，它可采用与主类相同约定。要注意的是类的名字必须是自主的。如果EMPLOYEE是领薪水的雇员和按小时工作的雇员的概化，那么恰当的名字是EMP、EMP_SALARIED、DEMP_HOURLY，或者是这些词的恰当的缩写。仅仅只有SALARIED类和HOURLY类是不够的，因为这些类中的每一个项都是一个雇员。

4. 属性的命名

属性采用和类一样的系统和缩写机制来命名，即描述属性的逻辑单词连接在一起。每一个属性都有一个后缀，用来标识域所属的种类，比如日期、号码、文本等。第6章会列出很多不同的域。利用这个列表，可以在属性名之后加上恰当的域后缀。这些后缀和它们的描述见表5-2。

在面向对象的方法中，布尔字段名通常加上“is”前缀，比如，活动标志命名为“isActive”。对这种语法的使用应加以注意。在这里，我们不选用这种方式，因为：

在Oracle中，不支持属性名中既有大写又有小写这种方式。“ISACTIVE”不像“isActive”那样易读。本应使用“IS_ACTIVE”，但是我们决定不这么做。

其实，我们喜欢在属性名之后加上后缀，这一点我们还想保持。

本应把这两种语法混用，即把这个字段称为“IS_ACTIVE_YN”，但是这样有点冗长了。

对每一个属性，曾考虑过增加一个缩写来表示实际的域，但实际上，文本长度和数字精度会经常改变。我们希望分类足够广，以使得改变很细微；但我们也想通过提供字段中有关数据类型的一些信息来辅助应用开发。通过使用这些后缀，可以建立一个工具，来辅助快速地为每一个属性分配恰当的域。如果把实际的域加在属性后面，那么在开发过程中，当决定改变域时，访问该属性的所有代码就也都会改变。在一个系统中属性长度的改变是很普遍的，但是属性从一种变到另一种却是很少见的。

第7章将提到，值列表类的主码称为<值列表类名>_CD。较长的值是<名字>_TX。如果采用系统生成的值列表类的主码，那么就要使用后缀ID来替代后缀CD。

表5-2 属性后缀

后 缀	描 述
-ID	数字型主码
-CD	字符型主码，以及来自于受限的字符值列表的任何属性
-YN	布尔型
-DT	日期，精确到日
-TM	日期，精确到秒
-TX	CD、YN之外所有其他的文本格式，包括 LONG 变量
-CY	代表通货数量的数字值
-NR	ID、CY之外所有其他数字字段
-GR	图象、图形、大型二进制对象

特定属性。有一些特定的属性需要稍微不同的命名约定，这些属性如表 5-3 所示。

表5-3 特定属性

属 性 名	描 述
ACTIV_YN	标识一个记录是否已经被删除的活动标记
ORDER_NR	用来同其他属性类区分的类的顺序号
DESC_TX	注解、评论等的长文本描述字段
_BY	特定类的系统用户名
CREAT_TM	记录生成的日期和时间
MOD_TM	记录最近修改的日期和时间
CREAT_BY	生成记录的系统用户名
MOD_BY	最近一次修改记录的系统用户名

5. 域的命名

对域名来说，主要要注意的是这些名字应足够短，使得它们能够填写到分配工具的小格里。就标准的域名而言，如ID或CODE，仅使用单词或恰当的缩写。

域名必须是描述性的。如果字段仅是特殊长度的域，则甚至可以使用实际的域，如 VARCHAR240。域的列表见第6章。

6. 关系的命名

这里我们主要探讨一下UML的命名标准。注意在ERD里，关系中只有两个事物需要命名，而在UML中，却有三个需要命名，即关系本身以及关系的双方。UML中使用的名字的语义同ERD中所使用的完全不同。ERD对关系的双方都使用动词短语，而UML只在一方使用动词短

语，此外还必须声明短语作用的方向。关系的名字以及约定的其他方面见第4章和第8章。

UML关系由一个动词和一个前置词组成的动词短语来描述，另外，还可以命名关系双方的角色。角色名使用的命名约定和类所使用的一样，角色名应清晰地阐明类中的对象在关系中所扮演的角色。图5-1描述了一个原型事例。

关系名的方向应该从高的基数到低的基数，必要时，只增加角色。

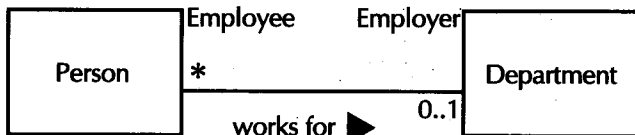


图5-1 显示关系名的UML图

7. 操作/方法的命名

作为“过程”（返回状态，不返回值）的方法必须用前缀 P_加上一个动词短语来命名。因为方法是明确属于类的，所以类名不需要是方法名的一部分。比如，在 EMP类中，方法“Hire”恰当的名字是“HIRE”，而非“HIRE_EMP”。作为“函数”的方法需要返回一个值，它必须命名为 F_<返回属性的名字>。以下给出了过程和函数名的例子。

过程的例子

Combine budgets

P_COMBN_BUDGT

Close Project

P_CLOSE_PROJ

函数的例子

Retrieve primary address

F_PRIM_ADDR_TX

Calculate total amount owed

F_TOT_AMT_OWED_NR

8. 描述创建

精确地描述一个类比给该类选择一个名字更为重要。描述一个主类可隐含地用短语“Each object in this class is a ...（类中每个对象是一个...）”开始。要了解类描述的更多细节，请参见第4章。

为开发者写的关于类的注释可放到描述成注释字段中。对关联类而言，经常要参照各个基本类。比如，要描述登记（Enrollment），它为学生（Student）和课程设置（Course Offering）之间的联系，类中的每一个对象都必须是“登记了一门特定课程的特定学生”。

值列表类的描述可用短语“Each object in this class is a valid value for...（类中的每个对象对...是一个合法值）”作为开始。这些值列表描述通常必须包含至少两个例子。

属性描述应该只在必要时才使用。很多属性都是很清楚的，不需要描述。例如，在地址（Address）类中的属性“城市（City）”就是如此，再建立描述就是浪费时间了。只在对本质的本质含糊不清的情况下才使用属性描述。例如，在 PERSON表中，可能会有一个属性叫“后缀（Suffix）”，这个属性的描述就是“表敬意的或其他性质的头衔，主要在写邮寄地址时附加在名字后面，如 MD、Ph.D.、III,等等。”

5.2 物理（实现）命名约定

在一个系统中，还有一些其他元素，它们不是逻辑数据模型的一部分，而是仅出现在物理实现过程中。这些元素也需要一致的命名约定。

1. 序列的命名

对系统产生的每一个主码列，我们都生成一个独特的序列。每一个表仅有一个序列，因而，序列名应是 <表名>_SEQ。例如，在 Project表中用来产生 PROJ_ID的序列就是 PROJ_SEQ。

2. 表名的短别名

为使名字简短，经常需要建立表名的短别名。为了生成这些别名，把类名中每个单词的头一个字母连接在一起。例如，PURCH_ORDER缩写为PO。如果这样不能保持名字的唯一性，就再增加单数字后缀，例如“PO”和“PO1”。

3. 约束的命名

如果在两个表中存在参照完整性约束，那么约束名就由 <第一个表名>_<第二个表名>_F组成。例如，如果在表 PURCH_ORDER和PURCH_ORDER_DTL之间有一个约束，那么约束名就是PURCH_ORDER_<PURCH_ORDER_DTL>_F。

对一个检查约束来说，命名约定应是

表名>_<列名>_C

如果这样太长的话，就使用没有双下划线的表名的短别名形式。如果这样还太长，那么就创建列的缩写。

对主码约束而言，命名约定应是

<类名>_PK

表5-4列出了约束名的一些例子。

表5-4 约束名的例子

类型	表	列	约 束 名
主码	EMP		EMP_PK
检查	MY_TABLE	MY_COL	MY_TABLE__MY_COL_C
检查	A_VERY_LONG_TABLE_NAME	MY_COL	AVLTN_MY_COL_C
检查	A_LONG_TABLE_NAME	A_LONG_COL_NAME	ALTN_ALCN_C

4. 索引的命名

索引命名采用下列格式：

<表名的短别名>_<索引中第一列的名字>_<索引中第二列的名字>_I

每个列名之间用双下划线隔开。如果这样太长，就采用表名的短别名缩写方法来缩写列的名字。若采用短缩写，就不需要双下划线。表 5-5列出了索引名的一些例子。

表5-5 索引命名约定的例子

表	索 引 列	索 引 名
EMP	DeptNo	EMP_DEPTNO_I
MY_TABLE	MY_FIRST_COL	MT_MFC_MSC_I
YOUR_TABLE	COL_ONE	YT_COL_ONE__COL_TWO_I
	COL_TWO	

5. 对象类型的命名

对象类型采用和类一样的命名约定，即在名字后加后缀 _T。

6. 过程和函数的命名

给过程加前缀P_，给函数加前缀F_。

7. 存储编程单元

存储编程单元是存储在Oracle数据库实例中的程序，执行一个或多个特定的任务。这些存储单元可通过用户接口如SQL*Plus来执行，也可在用Oracle Developer等建立的应用程序中引用。

可用的存储编程单元有三种类型：过程、函数和数据库触发器。“过程”是执行一个或多个动作的程序，它根据程序逻辑的完成/中断来选择性地返回值和/或状态信息。

存储在数据库中的所有过程和函数通常应该存储在包中，包有助于把过程和函数组织成逻辑组。所有的包都使用前缀为K_，以标识这些包后面跟的是一个用来描述包的单词。包名应该简短，因为任何对包内部过程或函数的引用必须有包名，调用时，包名加在最前面。如果包属于一个特殊表，包应取名为K_<短表名>；如果包属于一组表，就应选择一个比较短的单词来描述这组表，如K_ACCT可能是用于所有计帐函数的包，K_BUDG可能是用于所有与预算有关的函数。

当给包内的函数和过程命名时，要记住函数或过程的上下文是由包来决定的，这样名字通常可以缩写。比如，递交帐目事务的过程可称为P_POST_ACCT_TRAN，但因过程驻留在帐目（Accounting）包内部，所以它可称为POST_TRAN或者POST，这样可以很清楚地看出K_ACCT.POST的含义。

在给一个向表DEPT中插入对象的过程命名时，要在该过程作用的表“DEPT”的名字前加上前缀“P_”，结果可见代码5-4。

代码 5-4

```
--****reference example_05_01 as prerequisite
CREATE OR REPLACE PROCEDURE P_DEPT IS
BEGIN
INSERT INTO DEPT (DEPT_ID, NAME_TX, LOC_TX)
VALUES (95, 'ADVERTISING', 'HOUSTON');
END;
/
```

同样的方法也适用于存储函数。“函数”用来执行一项单独的特定任务，然后根据函数逻辑的完成/中断返回一个状态信息。过程和函数命名的主要不同在于代码5-5所示的前缀。

代码 5-5

```
--****reference example_05-02 as prerequisite
CREATE or replace FUNCTION F_CHECK_EMP_SAL (P_EMP_ID IN NUMBER)
RETURN NUMBER
IS
CURSOR C1 IS
SELECT SAL_NBR
FROM EMP
WHERE EMP_ID = P_EMP_ID;
FLAG NUMBER;
BEGIN
FOR C1_REC IN C1 LOOP
IF C1_REC.SAL_NBR < 5000 THEN
```

```
RETURN -1;  
ELSE RETURN 1;  
END IF;  
END LOOP;  
END;  
/
```

另一个值得考虑的、可以增加到过程和函数命名标准中的有用信息是 DML 指示器。即增加一个前缀，以“ I ”表示插入、“ U ”表示修改、“ D ”表示删除、“ S ”表示选择。这一部分显示出存储编程单元的主要用途。另外，这个标准会引导开发人员编写更加模块化的代码，从程序文档的角度来看，这一点是非常有用的。后面要讲的数据库触发器将说明这项标准的应用。

过程中的参数基本上采用同属性一样的命名约定，不同的是参数名多了一个前缀，用来标识它们在过程中的用法：

I_ 输入参数
O_ 输出参数
IO_ 输入/输出参数

8. 数据库触发器的命名

最后一个要说明的存储编程单元是数据库触发器。数据库触发器是隶属于特定表、根据特定行为准则执行的程序。数据库触发器基于对它所在的表的插入、更新以及删除请求而启动。

以下是用于数据库触发器的约定：

第一个字母是 B 或 A，表示“ Before ”或“ After ”。

第二、三、四个字母是 I（插入）、U（修改）、D（删除）或它们的任意子集。

例如，EMP 表中插入和修改操作之前激活的触发器称为 BUI_EMP；插入和删除操作之后（如果这种组合有意义的话）激活的触发器称为 AID。

可以指定这些触发器在执行它的 DML 语句（即 INSERT、UPDATE 或 DELETE 等）之前、之后、或者替代该 DML 语句执行。根据这个命名约定，触发器名的第一部分是一个字母——B 表示“ Before ”、A 表示“ After ”、I 表示“ Instead Of ”，接下来的三个字母表示引起触发器启动的 DML 行为。例如，假设表 DEPT 有一触发器，它在插入和修改之前启动，那么该触发器的代码如 5-6 所示。

代码 5-6

```
--***reference example_05_01 as prerequisite  
CREATE OR REPLACE TRIGGER BIU_DEPT  
BEFORE INSERT OR UPDATE  
ON DEPT  
FOR EACH ROW  
BEGIN  
NULL;  
END;  
/
```

注意，只有相关的 DML 类型（如触发器 BIU_DEPT 中的 I 和 U）才在触发器名字中出现。触发器名字中不会总是出现为 DML 操作设置的所有三个字母（I、U、D），而是依具体情况只出现相关的字母。