China-pub.com
下载

第12章　高级PL/SQL

PL/SQL是一种简单的语言，和许多简单的问题一样，简单中也有复杂性。特别情况时，使用第11章中介绍的工具可能创建出一个庞大的拙劣组织起来的代码结构，复制品再三地出现在混乱的应用程序中。

为了减小复杂性，PL/SQL提供了一系列工具来使代码系统变得简单一些。尽管这种结构增加了初始程序任务的复杂性，但是很快将发现，正确使用这些工具会使系统具有更好的可重用性，维护起来也更简单。这些工具包括异常错误的处理、组织大量代码的函数和过程、使用游标和其他数据结构把代码聚集为抽象数据类型的程序包，以及建立重用代码系统的库。

12.1　产生异常

在很多年的旁观之后，“程序异常”逐渐地找到进入主流数据库应用程序设计的道路。任何一个曾经编写过大型数据库应用的人都知道，数据库程序设计中的错误处理代码经常十分晦涩。在现代程序设计语言中，异常是进行错误处理的首选方式。可以用它来代替绝大多数GOTO语句，因为它在程序块中提供了容易理解的控制流程。

异常是程序识出或遭遇异常的条件时，运行时系统引发的一个 PL/SQL信号。这个信号事件有名字，并且有一个处理信号的系统：异常处理程序。

在程序设计中有三种异常可以使用：

- 内部PL/SQL异常：作为PL/SQL程序设计语言一部分的异常。
- 内部的运行时异常：作为 Oracle Developer运行时环境(Forms、Graphics和Reports都有不同的异常)一部分的异常。
- 用户定义异常：在代码中定义的处理Oracle错误信息或其他情况的异常。

回忆第11章的PL/SQL块的基本格式：

```
DECLARE
  --Data declarations
BEGIN
  null;--Program statements
EXCEPTION
  --Exception handlers
  WHEN OTHERS THEN
    null;--default handler
END;
```

本部分的重点就在这个程序块中最后的 EXCEPTION(异常)子句上。异常处理的结构与下面所示的子句类似：

```
EXCEPTION
WHEN <exception 1> THEN
  <statements>
...  -- more exception handlers
WHEN OTHERS THEN
```

China-pub.com
下载


```
<statements>
```

EXCEPTION子句由一系列异常处理组成，由 WHEN和异常名(exception name)开头。当程序块产生异常时， PL/SQL停止执行，程序块的EXCEPTION子句检查这个异常的异常处理。如果存在， PL/SQL跳转到该处理，执行后返回到调用者。如果不存在， PL/SQL跳出当前程序块，转到包含当前块的第一个块中的异常处理程序，也就是说，不执行程序块执行的主部分中进一步的程序语句。在产生异常和异常处理程序之间 PL/SQL不执行任何语句。

这种关闭异常的处理方法叫做异常的传播 (propagation)。如果顶层子程序不处理异常，控制传回到调用者的异常处理，并且异常在调用子程序中传播。最后，如果没有处理异常，则异常传播到宿主环境，进行默认处理，通常是回滚处理。 Oracle Developer应用程序将显示一个输出默认信息的警告窗口，然后进行适当的处理。例如，当产生 FORM_TRIGGER_FAILURE异常时， Oracle Developer Forms组件将终止触发器及其驱动的处理过程，通常把游标停留在引起异常的项或记录上。

可以直接通过RAISE语句产生一个任意的异常：

```
RAISE <exception-name>;
```

PL/SQL执行上面的语句时，则把控制转换到当前块的异常处理程序。如果块中没有处理这个异常，则传播异常。

如果要处理异常，但又不希望外部块或调用子程序发现异常，可以直接通过一条 RAISE语句传播异常：

```
RAISE;
```

这应该是异常处理的最后一条语句。

注意 如果在声明部分或异常部分产生异常，则异常立刻传播。所以必须把这个异常的任何异常处理程序放到一个外部块中。

下面的程序包定义了一系列嵌套的子程序，这些子程序将作为异常传播的例子。还在后面给出了程序包结构细节和使用。首先，必须创建程序包的说明：

```
CREATE OR REPLACE PACKAGE exceptionPackage IS
  eTest EXCEPTION;
  PROCEDURE OuterProc;
END exceptionPackage;
```

程序包用一系列嵌套的块和子程序实现 Outer_Proc过程。用块结构的复杂性举例说明异常是如何传播的。 Inner Block#2块出现一个test异常，并且异常通过异常处理程序传播。

```
CREATE OR REPLACE PACKAGE BODY exceptionPackage IS
  PROCEDURE OuterProc IS
  BEGIN -- OuterProc block
    DECLARE -- Start of Inner Block #1
      PROCEDURE InnerProc IS
      BEGIN -- Inner Procedure
        BEGIN -- Inner Block #2
          RAISE eTest; -- explicitly raise the exception here
        EXCEPTION  -- Inner Block #2 exception handlers
          WHEN eTest THEN
            Message('Exception handled in Inner Block #2');
            RAISE;
        END; -- End of Inner Block #2
```

China-pub.com 下载

```
    EXCEPTION  --Inner Proc exception handlers
      WHEN eTest THEN
        Message('Exception handled in InnerProc');
        RAISE;
    END InnerProc;
  BEGIN -- Inner block #1
    InnerProc;
  EXCEPTION  -- Inner Block #1 exception handlers
    WHEN eTest THEN
      Message('Exception handled in Inner Block #1');
      RAISE;
  END;  -- End of Inner Block #1
EXCEPTION  -- OuterProc exception handlers
  WHEN eTest THEN
    Message('Exception handled in OuterProc');
    RAISE;
END OuterProc;
END exceptionPackage;
```

下面的代码调用Outer_Proc过程：

```
BEGIN
  exceptionPackage.OuterProc;
EXCEPTION
  WHEN exceptionPackage.eTest THEN
    Message('Exception handled in calling procedure');
END;
```

在Oracle Developer中运行这段代码，产生如下输出信息：

```
Exception handled in Inner Block #2
Exception handled in InnerProc
Exception handled in Inner Block #1
Exception handled in OuterProc
Exception handled in calling procedure
```

Inner Block#2出现异常。其异常处理程序把异常传播到包含 Inner Block#2的InnerProc块中。InnerProc块的异常处理程序又将异常传播到包含它的 Inner Block#1块中,在Inner Block#1中，传播异常到所包含的 OuterProc。这个过程是分层结构的最顶层子程序，所以它把异常传播给调用子程序的块。

注意 当异常传播到子程序外部时，PL/SQL不设置子程序的OUT参数，并且自动地回滚子程序中完成的工作。如果不希望这样，则应该在子程序的外部块编写 WHEN OTHERS子句代码，来按照希望的方式处理异常。

12.1.1 PL/SQL的内部异常

PL/SQL定义了几种异常作为 STANDARD(标准)包的一部分。所有的 PL/SQL程序都能够引发这些异常。需要在代码中处理的公共异常是 NO_DATA_FOUND和VALUE_ERROR异常。当单行的选择 (SELECT…INTO)失败，返回一行时发生 NO_DATA_FOUND异常。VALUE_ERROR异常在语句出现值的问题 (算术、转换、截断或约束)时发生。产生 VALUE_ERROR异常的大多数情况是把过长的字符串赋给一个变量。例如，定义一个变量为 VARCHAR2(10)，并把字符串“ George E.Talbot ”赋给它，将引发 VALUE_ERROR异常。字符串有 14个字符，

China-pub.com
下载

而变量只允许有10个字符。

12.1.2 Oracle Developer的内部异常

Oracle Developer的三个组件都有其特定环境的异常。

Form Builder在正常运行处理中提供了FORM_TRIGGER_FAILURE异常。表单组件在处理的任何一个触发器失败时，产生这个异常。可以在触发器的代码中使用这个异常来告诉PL/SQL触发器失败：

```
RAISE FORM_TRIGGER_FAILURE;
```

Form Builder还提供了DEBUG.BREAK异常，可以在块中通过RAISE语句显式地嵌入调试中断。参见第15章。

Oracle Developer Reports组件把它的异常打包到SRW包中。

在Reports组件中两个最常用的异常是RUN_REPORT_FAILURE异常和PROGRAM_ABORT异常。RUN_REPORT_FAILURE异常表示报表失败，显示一个常规信息并结束报表。可以编写异常处理来显示用户自己的错误信息。在代码中出现致命的错误时，可以产生 PROGRAM_ABORT异常并显示适当的错误信息。

Oracle Developer Graphics组件有一大批内部异常，在此无法尽述。完整的列表参见联机的Graphics Builder参考。如果在Graphics Builder中大量地编写代码，将会发现要使用许多异常处理程序来处理意外情况的发生。在一个 Graphic的显示区有许多能够编程的对象，每个对象都可能有不同的错误发生。大约有 200种这样的异常。在使用 OG和TOOLINT包编程时可能会遇到它们。

例如，试图用OG_UPDATE_CHART显示日期坐标轴,但是日期超过了坐标轴的范围，Oracle Developer Graphics组件产生OG_DATE_OVERFLOW异常。当在坐标轴上指定一个无效的位置时，产生OG_INVALID_POSITION异常。

12.1.3 用户定义的异常

有两种用户定义的异常：Oracle错误异常和用户异常。

可以使用编译指令pragma EXCEPTION_INIT来为某个Oracle错误信息建立指定的异常。pragma是一条告诉编译器在编译时做什么的指令。既然这样，告诉编译器为 Oracle错误代码创建一个异常：

```
ePrivileges EXCEPTION;
PRAGMA EXCEPTION_INIT(ePrivileges, -1031);
```

这条语句把Oracle错误代码-1031与名字为privilegesException异常连接。当核心Oracle数据库服务器引擎发出这个错误代码给 PL/SQL时，PL/SQL产生这个命名异常并把控制传递到用户的异常处理程序。这个 PRAGMA保存某些代码。否则，需要捕捉 Oracle数据库服务器错误条件并为错误数(不需要编写的附加代码)测试错误代码(SQLCODE)。

要定义一个简单的用户异常，只需要在 Declare块中说明异常。例如，如果触发器为总帐记录做内部检验，则可能需要在检验失败时产生一个专门的异常，以便能够报告出这个问题：

```
DECLARE
  eValidation    EXCEPTION;
```

China-pub.com 下载

```
  vAlertButton NUMBER;
BEGIN
   null;  -- Do the validating.
EXCEPTION
  WHEN eValidation THEN
    vAlertButton := Show_Alert('LedgerValidationAlert');
    RAISE FORM_TRIGGER_FAILURE;
  WHEN OTHERS THEN
    RAISE FORM_TRIGGER_FAILURE;
END;
```

异常的作用域与变量相似。可以在不同的作用域中多次定义同一个异常。最好的方法是，仅仅在程序包中一次性定义一个给定的异常，将它导出到使用它的子程序中。可以用不同的名字定义异常，这可以确保处理异常的处理程序能够正确处理它。

注意 如果用同一个名字定义两个异常，则有两个独立的异常。这会使编写错误处理代码时非常混乱，尤其是在子程序有嵌套块时。用一个内置的异常名，如 NO_DATA_FOUND，来说明用户的异常尤其是不明智的。

12.1.4 使用块控制异常处理

在某些条件下，需要在处理异常之后继续原来的进程。但不幸的是，当 PL/SQL产生异常时，PL/SQL结束块的进程并在结束异常处理代码后返回调用者。这意味着不能返回块中引起异常的位置并且继续下去。通常的情况是，要在若干语句中取一个，或者希望即使在执行时由于某种原因引起失败，也要执行全部所有的代码。

如果想在处理过程中有完全的控制权，则必须把子程序分开到嵌套块中。关于块嵌套的详细信息，参见下面关于子程序(过程和函数)部分。需要做的是把每个“不致命”的异常的异常处理程序放到单独的块中。某个异常处理程序执行之后，对于嵌套的块仍然存在并继续执行，只是跳过了嵌套块中的代码。

例如，要浏览一个表中的一个数据，但也希望在原始值不存在时建立一个默认值。需要执行几条SQL语句来确定真实状态，并根据情况分支，结构如下：

```
DECLARE
  vLookupValue VARCHAR2(100);
BEGIN
  BEGIN
    SELECT lookupValue INTO vLookupValue FROM LookupTable;
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      vLookupValue := 'N/A';
  END;
  INSERT INTO AuditTable (currentDate, currentValue)
    VALUES (sysdate, vLookupValue);
END;
```

这个块首先执行单行选择来发现查找值。如果服务器没有返回数据，嵌套块异常处理程序赋了一个默认值，并且继续处理在嵌套块中 END下面的第一条语句：INSERT语句。

12.1.5 错误处理函数

有两个PL/SQL内部函数可以在异常处理程序中处理数据库的错误。

China-pub.com
下载

SQLCODE函数返回最后错误的错误代码。这个代码对于在 PRAGMA EXCEPTION_INIT中定义的错误来说，是一个负数，或者是 +100(NOT FOUND)，或者是+1。

SQLERRM函数是错误信息文本，包含错误代码。

Oracle Developer Forms也提供了一些错误处理内部函数，见表 12-1。

表12-1 Oracle Developer Forms组件的内置错误处理函数

函 数	返回值类型	描 述
ERROR_TYPE	CHAR	返回最后发生的错误的类型：Forms错误为FRM，数据库错误为ORA
ERROR_CODE	NUMBER	返回最后Forms错误的代码
ERROR_TEXT	CHAR	返回最后Forms错误的文本
DBMS_ERROR_CODE	NUMBER	返回Forms检测到的最后数据库错误的代码
DBMS_ERROR_TEXT	CHAR	返回一个或多个数据库错误消息的类型和文本，在其前面是DBMS_ERROR_CODE

12.2 减少使用匿名块

第11章中采用的无名块对触发器建立简单大块代码有很大影响。当需要使用可重用代码时，无名块变得用处不大。应该到更正式的命名块和参数化块的组织机构去进一步讨论。

12.2.1 子程序

PL/SQL提供了两种实现语言组织功能的对象：过程和函数。这两种代码组织都是子程序：是一些有名字的、参数化的块代码，可以用参数来调用它们执行某些操作。函数是计算某个值的子程序，而过程只是执行一个动作。在 Oracle Developer中，可以用它们在程序单元标题下面的各种组件中编写代码，或者把它们添加到库组件中。必须在触发器代码中或其他任何引用子程序名的地方显式地调用子程序 (例如，在报表组件中的组过滤器和公式中使用函数)。调用子程序是一个 PL/SQL语句。因此，在建立一个子程序时，是通过增加一种新语句扩展了PL/SQL语言。

通常，在需要计算一个单一值时可以编写函数。在想执行一个操作而不是计算一个值时，应该编写一个过程。然而，所有的子程序都可以有输出参数。因为希望函数返回一个单一值，所以在编写函数代码时不应该使用 OUT或IN OUT参数。对于错误条件和返回及其他类似操作，应该加以定义并截获异常 (见前面的部分)。有关输出参数的更多信息见下面部分。

子程序可以重载。可以在程序包中或另外子程序中或有多个子程序的块中用同样的名字声明多个子程序。细节见本章 12.3节 “ 创造性的包 ” 部分。

1. 过程

过程如下所示，由名字、一组参数和一段代码组成：

```
PROCEDURE <name> [<argument list>] IS
  <declarations>
BEGIN
  <program statements>
END;
```

除了用 PROCEDURE…IS序列代替了 DECLARE 关键字之外，其语法与无名块相同。PROCEDURE…IS序列是过程说明，其余的部分是过程体。可以在调用子程序的子程序或块

China-pub.com 下载

之前只编写说明代码，然后在后面的代码中编写完整的子程序。

2. 函数

函数由名字、一组参数、返回值类型和一段代码组成。当从函数返回时，函数返回给调用者一个指定类型的值(这种语法与过程的相似，除了说明中的RETURN子句)。

```
FUNCTION <name> [<argument list>] RETURN <type> IS
  <declarations>
BEGIN
  <program statements>
END;
```

为了完成函数的处理并返回函数值，使用的 RETURN语句应该包含为返回值赋值的表达式。

注意 如果你高兴可以有很多个RETURN语句，但是这是一个坏习惯，RETURN语句最好不要超过一个。有两个或更多的RETURN语句将导致程序控制的可能路径增多，使程序单元的测试变得困难。当进行集成测试时，必须去检验每个调用者的返回值以确定调用者取得了正确值。因此，多个返回节省了一些代码，却导致测试工作量和潜在缺陷的巨大增加。可以在子程序中使用内部异常，并从异常处理程序返回。一般来说，尽管如此，在编写代码时最好避免这种情况。可以通过嵌套块来实现这个目的 (参见12.2.2节“嵌套块”)。

3. 参数

通过在括号中的一个说明列表来为子程序定义一组参数：

```
( <variable> [ <mode> ] <type> [ := | DEFAULT <value>, ... )
```

<variable>是标准PL/SQL标识符，表示参数名。<mode>是如下三种可能性的一种：

- IN：这个参数的值是子程序的输入值，给该变量赋值不影响其在调用程序中的值。如果不指定方式，这是默认的方式。这种参数可以是变量、常量、直接量或表达式。
- OUT：这个参数用于从子程序中输出，该变量用来赋一个子程序返回时调用者能够使用的值。在没有赋值之前，不能使用这个变量。该变量在调用者中必须是能够修改的变量，并且必须是变量而不是直接量，因为不能给一个值赋值。
- IN OUT：这个参数是既能用于输入又能用于输出的变量。可以使用其输入值，又能给它赋一个调用者能够看到的输出值。因为要给参数赋值，这种参数必须是变量。

另外，在函数中应该避免使用OUT或IN OUT参数，因为函数通常只返回一个唯一的返回值。

<type>可以是任意的有效PL/SQL类型，包括Oracle Developer对象或自定义的类型。不需要指定类型的刻度或精度，例如VARCHAR2(20)，只使用类型名。

“:=IDEFAULT<value>”子句表示既可以使用赋值操作 :=，也可以使用DEFAULT关键字给参数赋一个默认值。这可以使得可以不带参数而调用子程序，在这种情况下，子程序给参数赋默认值。这种功能只能用于IN变量，不能用于OUT或IN OUT变量。

列表中参数之间用逗号分隔。

如果子程序没有参数，则该子程序没有括号：

```
PROCEDURE <name>;
FUNCTION <name> RETURN <type>;
```

China-pub.com
下载

4. 调用子程序

子程序调用是一条PL/SQL语句，可以在放置PL/SQL语句的任何地方放置这样一条调用语句。调用子程序只需要块中的一条语句：

```
Do_Key('NEXT RECORD');
```

可以把子程序的调用作为无名块的一部分或过程的一部分：

```
DECLARE
  vAlertButton NUMBER;
BEGIN
  vAlertButton := Show_Alert('WarningAlert');
END;
```

提示 当调用一个函数时，如前面例子所示，必须把返回结果赋值给一个变量。如果不这样，将出现PL/SQL找不到过程的提示。PL/SQL如何确认一个函数与返回类型无关。

在哪里说明的子程序决定了在哪里能够调用它。如果在一个模块 (表单、菜单、报表或图形显示)中定义了一个子程序，只有这个模块的触发器和其他子程序能够调用这个子程序。例如，如果在表单模块中定义了一个过程，只有表单的触发器和表单子程序能够调用这个子程序。如果在菜单模块中定义了一个函数，只有在菜单项命令和初始化代码中能够调用这个函数。

如果在库模块中定义了一个子程序，必须首先把库连接到模块，才能调用这个子程序。为了实现这个目的，在模块中找到 Attached Libraries标题，单击Create按钮，通过结果对话框选择连接库。

调用存储的子程序(在服务器上定义的子程序，而不是在库或模块中)稍微复杂一点。一个好方法是，首先为子程序定义一个公共同义词。有了这个同义词，调用者只须指定子程序的名字即可。

另外，必须有适当的权限来执行过程。如果用过程的所有者身份登录，则自动取得EXECUTE(执行)权限，所有者必须给你的用户或角色授予这种权限：

```
GRANT EXECUTE ON Talbot.AddLedgerTransaction to AccountingRole;
```

现在，任何被授权AccountingRole角色的用户都可以执行AddLedgerTransaction子程序。

注意 如果在调用子程序，且PL/SQL不能识别子程序名，应该仔细检查过程说明的类型和参数个数。PL/SQL使用子程序名、返回类型(对函数而言)和参数的存在及类型来识别一个特定的子程序。如果有一项不存在，或者指定的类型不正确，或者传递的参数类型不正确，则PL/SQL不能确认对被调用子程序的调用。如前面所述，如果使用一个函数，要确定把返回值赋值给了一个变量，因为PL/SQL使用它来确认函数名。

按位传递 传递参数给子程序有两种途径。第一种途径是按位传递 —— 按子程序定义参数的顺序传递参数值。例如，说明过程如下：

```
PROCEDURE testLedgerItem(pItemID NUMBER, pAmount NUMBER);
```

过程调用如下：

```
testLedgerItem(456, :Ledger.Amount);
```

值“ 456 ”传递给pItemID，表单项Ledger.Amount传递给pAmount。注意，传递项目对象值(item)使用绑定变量的语法(冒号前缀)。

按名传递 另外一种途径是按名传递。使用这种语法，可以按任意参数名的顺序传递参

China-pub.com 下载

数值：

```
testLedgerItem(pAmount => :Ledger.Amount, pItemID => 456);
```

按名传递对默认值尤其有用。如果在有默认值时使用按位传递，可以省去有默认值的尾随参数。如果使用按名传递，可以省去任何有默认值的参数，不仅仅是尾随参数。这意味着，可以省去某些有默认值的参数，但要包含其他在参数定义中没有默认值的参数。

5. 报表过滤器和公式

在Oracle Developer Reports中函数的一个主要用途是实现过滤器和公式，细节参见第 7章。

报表组过滤器是一个返回 Boolean值的函数，Oracle Developer用它来决定在组中显示哪些记录。通过设定组过滤器属性的 Condition(条件)和提供一个函数，告诉 Reports只显示组中那些函数值为TRUE的记录。

报表公式是一个函数，它利用类型属性公式为分组中的每个记录返回一个值，并附加到一个数据列中。可以使用一个表达式来代替这个函数，但是那样就不能使用函数所使用的PL/SQL的完整功能。函数返回一个与列数据类型项对应的单一值：数值列的函数返回值为NUMBER类型；日期列的函数返回值为 DATE类型；字符串类函数的返回值为 CHARACTER、VARCHAR或VARCHAR2类型。

12.2.2 嵌套块

PL/SQL的块结构本性提供了在作用域方面对子程序部分的强大控制。程序的块结构是使用PL/SQL块(DECLARE…BEGIN…END;序列)来组织变量有效性和异常处理程序的途径。

块结构的基本概念是一个块是一条程序语句。这条语句按顺序执行，像任何程序语句一样。然而，在语句中可能有与外部块及其进程不知道的事情有关的庞大的复杂结构。隐藏 (或者封装)可以使内部块的复杂工作与外部块的顺序无关。这样减小了程序的复杂性，使程序更容易理解、调试、测试和维护。例如，可以推迟嵌套块体的代码编写，使程序残留一些不完全的地方，然后可以继续编写代码和测试子程序的其他部分。在需要用到嵌套块之前，不需要考虑嵌套块的细节。

在前面关于异常处理的部分中提供了一个使用嵌套块控制异常处理结束之后执行流程的例子。通过把异常处理程序放到一个内部块中，可以实现在异常处理之后继续外部块的处理。

块结构也能够在子程序中隔离代码的逻辑簇。例如，需要一个有多个返回的子程序，则应该考虑嵌套那段不想执行的代码，并且有条件地执行那个块。然后子程序就有唯一的退出点。这可以使程序的控制结构清晰，尤其对于测试有用。

通过说明嵌套块，可以在块间共享数据元素而不需要使用全局变量。嵌套块可以访问外部块说明的所有变量。

可以通过在外部子程序说明部分包含子程序，把子程序整个嵌套到其他子程序中。必须在所有的其他说明(变量、类型或异常)之后声明嵌套子程序。

12.2.3 在导航器中定位块

现在知道了如何编写子程序。下面的问题是在什么地方、如何编写子程序。前面讲述了两个可以编写子程序的地方：

■ Program units(程序单元)：在表单、菜单、显示或报表模块的 Program Units标题之后。

China-pub.com
下载

■ Library(库)：在Forms、Reports或Graphics的库中Libraries 标题之后。

也有第三种选择：

■ Stored(存储)：通过CREATE FUNCTION和CREATE PROCEDURE语句或者通过在Database Objects标题下建立子程序存储在数据库中。

如何确定在哪儿定义子程序？

如果使用子程序访问那些不希望普遍可用的数据，则使用存储子程序。这是个封装过程，在6.2.5节“过程的封装”中讨论。如果在子程序中完成许多数据库工作，也使用存储子程序。这可以使调用子程序和服务器执行每件事更有力 (大概)。这也可以减少网络传输。如果操作是对数据库对象的常规操作，使用存储子程序。这可以与对象的所有用户共享操作。这是结合性原则，把运算存储到其操作的数据附近。

注意 因为服务器和客户机版本的 PL/SQL可能有差别，可以建立用服务器端版本PL/SQL封装函数的存储过程供客户端版本使用。例如，如果对Oracle 7和Oracle 8使用Oracle Developer，就不使用同样版本的PL/SQL处理。可以使用DBMS_SQL建立存储子程序，适当地使其参数化，然后从客户端PL/SQL调用它们。当运行Oracle 7数据库时，在服务器上运行Oracle 7 PL/SQL；当运行Oracle 8数据库时，运行PL/SQL8。

如果想要使子程序被多于一个的模块使用，则使用库。这通常适用于适应不同情况的常规操作。另外，子程序应该只在几个有限的方式下适用于数据库。存储子程序用于数据库相关操作很多的情况下，可以减小网络传输和使得重使用更加容易。有一种情况例外，这就是子程序需要调用组件的内部子程序。因为无法从数据库服务器调用 Oracle Developer内部子程序。另外一种例外情况是子程序需要引用 Oracle Developer模块的变量。然而，可以把这些值作为参数传递给子程序，而不是直接引用它们。这种子程序和数据模块的松耦合是一个优良的系统结构的基本原则。这种送耦合意味着可以把子程序存储到数据库并把组件模块传递给它。

本章最后的“检查库”节讲述了如何在库中建立子程序。一旦编写了子程序代码，就可以把库附加到任何模块并在模块中使用子程序。要附加一个库，在模块中找到 Attached Libraries(附加的库)的标题，选择它，单击Create工具。然后就可以通过一个标准的 Open File对话框选择附加库。不要忘记了从文件名中移除所有的路径信息。

子程序在模块中。当代码对模块十分特殊，在其他模块中没有任何用处时，应该把子程序放在模块中。模块大量使用项、大量使用应用程序专用的单独逻辑关系、通过内部子程序对其他组件的大量使用，这些都表明模块的特殊性。

要建立一个模块程序单元，在模块下找到 Program Units(程序单元)的标题，单击Create工具，会看到到一个 New Program Unit对话框，使用同样的 PL/SQL Editor定义子程序。但是Oracle Developer分别用表单、菜单、显示或报表模块存储代码。后面的“建立库”部分详细讲述了如何使用这些工具建立子程序和包。

另外一种情况是定义包含子程序操作的对象组。这种对象组的行为好像是一种程序包，它使得在程序包中的子程序在其他模块中的重用有意义。这是 PL/SQL包或库的选择对象，在后面的部分将介绍更多的内容。

当Oracle Developer寻找程序单元时，首先检查模块的 Program Units部分。其次，在模块中Attached Libraries部分连接的库的程序单元中查找。第三，在数据库服务器上查找。使用这

China-pub.com 下载

个顺序的优点是有助于在分级结构的高层使用同样的名字添加程序单元。例如，有一个库子程序支持大多数应用程序的默认行为。在模块的 Program Units部分添加了一个与其名字一样的子程序。当Oracle Developer运行应用程序并调用子程序时，则首先查找添加的子程序，决不会执行库子程序。

12.3 创造性的包装

程序包可以将其他PL/SQL元素联合到一个独立的整体中：一系列数据对象、类型、游标、异常和子程序。程序包有说明和程序体。程序包的说明是它的公用面：说明它所使用的元素。这些元素是公用元素。包的程序体是游标和所定义的子程序的实施，也包含要说明的私有元素。可以替换包程序体而不改写包的说明。

注意 不能在程序单元中嵌套程序包。

12.3.1 创建程序包

为什么要创建程序包？程序包提供了一种方式来将 PL/SQL元素结合到一个系统中。程序包与对象定向系统的对象相比：程序包中包含与系统中某些结合的部分相关的状态和行为。程序包提供了完整的数据抽象和信息隐蔽，从而提高了系统的模块性。这种改善了的内聚性和数据抽象之间的松耦合，又直接提高了代码的可读性和软件的维护能力。另外，一旦要执行程序包中的模块，Oracle将把程序包加载到内存中。所以使用程序包组件的进一步处理会执行得更加好。

例如，可以改变程序包中的函数而不用改变任何调用这个函数的调用者，因为接口说明没有改变。也可以使一个子程序成为存根程序 (stub)直到为它们编写代码为止，甚至可以根本不为其提供程序体。这可以在程序包还没有真正完成的情况下对使用程序包的程序单元进行编译，进行可能的单元测试。

程序包也能够定义变量、常量和在整个会话中持续的游标为公有元素或私有元素。提供了在过程调用或事务处理之间保持信息而不依赖于无特定结构的全局变量的能力。通过程序包，可以使数据和操作的控制结构化。所有这些元素都是 NULL ，除非在说明中初始化它们。可以定义子程序完全在程序包的程序体中。那样，就不能在程序包以外访问这些子程序。

最后，当使用程序包的任何一个部分时，PL/SQL的程序包技术加载整个程序包到内存中。这意味着对程序包中程序另外的调用不需要另外加载代码，提高了性能。这个功能也说明，程序包要相对简单并且要内聚，而不是把一大堆不相干的代码集合到一个程序包中。

有三种内聚表明应该使用程序包：

- 功能内聚：一组元素作为一个整体，描述应用的一个特定的功能，例如一个对象类型或与簇相关的对象类型。
- 抽象内聚：这是一组程序单元，它们作为提供一组定义明确的服务的抽象层在一起工作。一个应用程序界面或 API(应用程序接口)表现出这种结合，其例子包括 Oracle Report Builder的SRW包和Graphics Builder的OG包。
- 实用性内聚：在一起使用的一组对象与 /或程序单元，在其中没有服务或功能的使用，是实用性结合。一个特殊的实用性内聚的情况是事务处理及其元素 (无论如何不同)打包到一个游标和子程序包中。

China-pub.com
下载

定位程序包的逻辑与子程序相同 (如果必要，回顾本章前面“在导航器中定位块”部分)。如果程序包结构建立得好，很容易区别数据库相关程序包和纯粹的客户端定向程序包。由于重用性的缘故，大多数程序包应该在库中开发，而不是在模块中开发。有一个例外是像块或图表那样的模块对象聚合一个程序包。建立程序包的过程与子程序的建立完全相同。参见本章“子程序”部分。

程序包有两种语法，一种是程序包说明语法，另外一种是程序包程序体语法。

1. 程序包说明

```
PACKAGE <name> IS
  {<variable spec>|<type spec>|<cursor spec>|<exception spec>} ...
  <subprogram spec> ...
END;
```

关于变量和类型说明的细节参见第 11章。说明包括一组对象说明 (变量、常量、类型、游标和异常)和一组子程序说明。

游标按如下格式说明：

```
CURSOR <name> <parameter list> RETURN <type>;
```

游标说明，因为不包括SELECT语句，必须提供RETURN子句来识别游标取回记录的类型。<type>可以是下面的任何一种类型：

- Record：先前定义的记录类型。
- Variable%TYPE：先前定义的变量类型。
- Table.Column%TYPE：数据库列类型。
- Table%ROWTYPE：一个表中所有列的记录类型。

如果要做的不仅仅是从表中选择列，例如 SELECT序列中包含GROUP BY或者表达式，则必须提供记录类型。这个类型必须与程序包程序体中游标体中定义的类型相匹配。游标说明的目的是定义游标的接口，游标接口由游标名、参数列表和返回类型组成。

子程序说明语法如下：

```
PROCEDURE | FUNCTION <name> <parameter list>
  [RETURN <type>];
```

参数列表的语法参见前面的“参数”部分。 RETURN子句只在函数中提供。在一个完整的子程序定义中，语句在IS和子程序体之后继续。子程序说明的目的是导出子程序的接口，程序的接口由子程序名、参数列表和返回类型组成。在程序包说明中，子程序的说明在其他说明之后。在程序包中子程序可以重载，可以在程序包中或另外的子程序中或块中用同样的名字说明多个子程序，只要参数的数目与 /或参数类型不同。这可以建立通过不同参数可选择的子程序。一个例子是Oracle Developer内部程序包中的内部子程序，其第一个参数既可以是对象名，也可以是对象标识符 ID。在内部程序包 (Forms STANDARD、Reports SRW或Graphics OG)中确实有两个独立的、重载的子程序。子程序的重载可以为不同用户建立一组非常灵活的操作，使程序包的可重用性更大，使用更容易。

注意 类型必须完全不同。REAL和NUMBER及FLOAT类型在考虑子程序重载时，它们是相同的。这适用于函数的RETURN类型。

下面程序包说明的例子是处理 24小时时间值的程序包的接口说明。程序包包含一个记录类型、一组异常和对记录类型操作的算术函数。这个程序包引用另外一个程序包 TimeInterval-

China-pub.com 下载

Package，这个程序包导出一个时间间隔类型 (TimeIntervalType)和处理这种类型的函数。为了方便指定的时间和时间间隔，按任意先后顺序相加，重载了算术加法运算符。减法是不对称的。

```
CREATE OR REPLACE PACKAGE Time24Package IS
  TYPE tTime24 IS RECORD (
    vHour NATURAL NOT NULL := 0,
    vMinute NATURAL NOT NULL := 0,
    vSecond NUMBER NOT NULL := 0.0);
  eInvalidHour EXCEPTION;
  eInvalidMinute EXCEPTION;
  eInvalidSecond EXCEPTION;
  FUNCTION CurrentTime RETURN tTime24;
  PROCEDURE SetTime(pTime IN OUT tTime24, pHour IN NUMBER,
    pMinute IN NUMBER, pSecond IN NUMBER);
  FUNCTION Add(pTime IN tTime24,
               pInterval IN TimeIntervalPackage.tTimeInterval)
    RETURN tTime24;
  FUNCTION Add(pInterval IN TimeIntervalPackage.tTimeInterval,
               pTime IN tTime24)
    RETURN tTime24;
  FUNCTION Subtract(pTime IN tTime24,
                    pInterval IN TimeIntervalPackage.tTimeInterval)
    RETURN tTime24;
  FUNCTION Format(pTime IN tTime24) RETURN VARCHAR2;
  PROCEDURE Validate(pTime IN tTime24);
END Time24Package;
```

这是描述一个对象的程序包的例子。 TYPE说明不建立真实记录，只定义记录的类型。函数都引用了一种或两种记录类型。程序包自己本身不包含数据。在使用程序包的子程序中建立数据对象、记录类型变量。然后调用与该变量有关的函数完成对变量的操作，或者通过返回一个新的对象类型建立新数据。记录的个体元素作为说明的一部分对于调用者是可见的。忽略它而对数据操作只使用函数是一个好的习惯。

注意 在内存中程序包只存在一份。这表示如果把变量作为程序包的一部分定义，那么只有一个这样的变量。如果这能够满足需要，那最好了。上面的例子很清楚地表明在那里它是不合适的。有可能不得不建立多种时间值，而不仅仅是一个。这意味着不能完全把记录类型结构封装到程序包中——至少是没有大量的程序编制、间接处理、艰难的维护。这是一个在哪里使用比较好而不是坚持“按正确的方式去做”的事例。可以尽可能容易和少付出努力去做想要做的事，除了冒滥用权限直接访问结构的危险之外。

2. 程序包程序体

程序包程序体语法如下：

```
PACKAGE BODY <name> IS
{<variable spec>|<type spec>|<cursor body>|<exception spec>} ...
  <subprogram body> ...
[BEGIN
  <statement>; ...
[EXCEPTION
  <exception handlers> ...]]
END;
```

China-pub.com
下载

只有游标和子程序体与在程序包说明中说明的子程序相对应。其余的说明 (变量等等)对于程序包的程序体都是私有的，不能在外部代码中使用这些元素。

游标体的语法与游标说明的语法相似，除了附加的 SELECT语句：

```
CURSOR <name> <parameter list> RETURN <type> IS <SELECT statement>;
```

参数列表必须与游标说明中的参数列表精确匹配， RETURN类型也是这样。

程序包程序体初始化在程序包程序体的末尾随意的 BEGIN…END序列之间，在说明部分定义完所有游标和子程序元素之后。这块代码只在第一次引用程序包元素时执行一次，使用它来初始化程序包的私有或共有元素。异常处理程序处理初始化代码引起的任何异常。

下面的例子显示了前面部分定义的程序包的程序体：

```
CREATE OR REPLACE PACKAGE BODY Time24Package IS
  cSeparator CONSTANT CHAR(1) := ':';  -- format cSeparator hh:mm:ss
  cSecondModulus CONSTANT NUMBER := 60.0;
  cMinuteModulus CONSTANT NATURAL := 60;
  cHourModulus CONSTANT NATURAL := 24;
  cSecondsPerMinute CONSTANT NUMBER := 60.0;
  cSecondsPerHour CONSTANT NUMBER := 60*cSecondsPerMinute;
  cSecondsPerDay CONSTANT NATURAL := 24*cSecondsPerHour;
  eSecondsOverflow EXCEPTION;  -- internal error

  -- An internal Convert function to convert time values
  -- to and from seconds
  FUNCTION Convert(pTime tTime24) RETURN NUMBER IS
    vSeconds NUMBER := 0;
  BEGIN
    Validate(pTime);
    vSeconds := (pTime.vHour * cSecondsPerHour) +
                (pTime.vMinute * cSecondsPerMinute) +
                 pTime.vSecond;
    IF vSeconds NOT BETWEEN 0 AND cSecondsPerDay THEN
      RAISE eSecondsOverflow;
    END IF;
    RETURN vSeconds;
    EXCEPTION
    WHEN eSecondsOverflow THEN
      DBMS_Output.Put_Line('Error converting seconds, number out of
range:  '||
                To_Char(cSecondsPerMinute));
      RETURN 0.0;
  END Convert;

  FUNCTION Convert(pSeconds NUMBER) RETURN tTime24 IS
    vTimeBuffer tTime24;  -- return value
    vSecondsBuffer NUMBER := pSeconds;
 BEGIN
   IF pSeconds > cSecondsPerDay THEN
     RAISE eSecondsOverflow;
   END IF;
   vTimeBuffer.vHour := FLOOR(vSecondsBuffer / cSecondsPerHour);
   vSecondsBuffer := MOD(vSecondsBuffer, cSecondsPerHour);
   vTimeBuffer.vMinute := FLOOR(vSecondsBuffer / cSecondsPerMinute);
   vSecondsBuffer := MOD(vSecondsBuffer, cSecondsPerMinute);
   vTimeBuffer.vSecond := vSecondsBuffer;
   Validate(vTimeBuffer);
```

China-pub.com 下载

```
    return vTimeBuffer;
  EXCEPTION
    WHEN eSecondsOverflow THEN
      DBMS_Output.Put_Line('Error converting seconds, number too large:
'||
              To_Char(cSecondsPerMinute));
      SetTime(vTimeBuffer, 0, 0, 0);
      RETURN vTimeBuffer;
  END Convert;

  FUNCTION Add(pTime tTime24,
               pInterval TimeIntervalPackage.tTimeInterval)
    RETURN tTime24 IS
    vTimeBuffer   tTime24;  -- Buffer for return value
    vCarry  NATURAL := 0;  -- carry digits to next component
  BEGIN
    Validate(pTime);
    TimeIntervalPackage.Validate(pInterval);
    vTimeBuffer.vSecond := MOD(pTime.vSecond + pInterval.vSecond,
                               cSecondModulus);
    vCarry := FLOOR(pTime.vSecond + pInterval.vSecond /
cSecondModulus);
    vTimeBuffer.vMinute := MOD(pTime.vMinute + pInterval.vMinute +
vCarry,
                               cMinuteModulus);
    vCarry := FLOOR(pTime.vMinute + pInterval.vMinute + vCarry /
                    cMinuteModulus);
    vTimeBuffer.vHour := MOD(pTime.vHour + pInterval.vhour + vCarry,
                             cHourModulus);
    Validate(vTimeBuffer);
    RETURN vTimeBuffer;
  END Add;

  FUNCTION Add(pInterval TimeIntervalPackage.tTimeInterval,
               pTime tTime24)
    RETURN tTime24 IS
  BEGIN
    RETURN Add(pTime, pInterval);
  END Add;
FUNCTION Subtract(pTime tTime24,
                  pInterval TimeIntervalPackage.tTimeInterval)
    RETURN tTime24 IS
  vTimeBuffer tTime24;  -- Buffer for returned value
  vSeconds1 NUMBER := 0.0;
  vSeconds2 NUMBER := 0.0;
  vResultSeconds NUMBER := 0.0;
BEGIN
  -- Convert the time and interval to seconds to simplify
  vSeconds1 := Convert(pTime);
  vSeconds2 := TimeIntervalPackage.Convert(pInterval);
  vResultSeconds := vSeconds1 - vSeconds2;
  IF vResultSeconds NOT BETWEEN -cSecondsPerDay AND cSecondsPerDay
    THEN RAISE eSecondsOverflow;
  ELSIF vResultSeconds < 0 THEN
    -- Add negative value to total seconds per day to get the
    -- "reverse" number of seconds
    vResultSeconds := cSecondsPerDay + vResultSeconds;
```

China-pub.com
下载


```
    END IF;
    vTimeBuffer := Convert(vResultSeconds);
    Validate(vTimeBuffer);
    RETURN vTimeBuffer;
  END Subtract;

  FUNCTION CurrentTime RETURN tTime24 IS
    vDateTime DATE := SYSDATE;
    vTime tTime24;  -- return value
  BEGIN
    vTime.vHour := To_Number(To_Char(vDateTime, 'HH24'));
    vTime.vMinute := To_Number(To_Char(vDateTime, 'MI'));
    vTime.vSecond := To_Number(To_Char(vDateTime, 'SS')); -- no
fraction
    Validate (vTime);
    return vTime;
  END CurrentTime;

  PROCEDURE SetTime(pTime IN OUT tTime24, pHour NUMBER, pMinute NUMBER,
                    pSecond NUMBER) IS
    vTimeCopy tTime24;
  BEGIN
    pTime.vHour := pHour;
    pTime.vMinute := pMinute;
    pTime.vSecond := pSecond;
    vTimeCopy := ptime;
    Validate(vTimeCopy);
  END SetTime;
  -- 1 140
   FUNCTION Format(pTime tTime24) RETURN VARCHAR2 IS
   BEGIN
     Validate(pTime);
     RETURN To_Char(pTime.vHour)||cSeparator||
            To_Char(pTime.vMinute)||cSeparator||
            To_Char(pTime.vSecond);
   END Format;

   -- This internal procedure validates the time record
   PROCEDURE Validate(pTime IN tTime24) IS
   BEGIN
     IF pTime.vHour NOT BETWEEN 0 AND 23 THEN
       RAISE eInvalidHour;
     END IF;
     IF pTime.vMinute NOT BETWEEN 0 AND 59 THEN
       RAISE eInvalidMinute;
     END IF;
     IF pTime.vSecond NOT BETWEEN 0 AND 59.0 THEN
       RAISE eInvalidSecond;
     END IF;
   END Validate;

END Time24Package;
```

注意 对程序包有某些约束。首先，当调用程序包子程序时，Oracle执行一个隐含的保存点(savepoint)。如果子程序失败并有未处理的异常，PL/SQL在调用者产生异常之前，回滚事务处理到这个保存点。第二，如果在分布式事务处理中使用程序包，不能包含任何事务处理命令(COMMIT、ROLLBACK或SAVEPOINT)。

China-pub.com 下载

12.3.2 使用程序包

通过用程序包名、圆点分隔符和元素名组成的格式来引用程序包元素：

```
DECLARE
  vTime Time24Package.tTime24;
BEGIN
  vTime := Time24Package.CurrentTime;
END;
```

注意 如果使用服务器上的存储程序包，必须要有程序包的EXECUTE权限。和子程序不同，必须把EXECUTE权限赋给用户，而不是把一个角色(role)赋给用户。另外，定义公用同义词是一个好方法，同义词将使应用能够引用程序包，不须在程序包名前加上用户名前缀。

12.3.3 内部程序包

Form Builder的几个内部程序包提供了对应用程序很好的控制。尽管这些程序包超出了本书的讨论范围，但仔细地查阅这些程序包的参考手册的内容会对在应用中任何大量的 PL/SQL编程有好处。

Form Builder提供的程序包见表 12-2。程序包显示在 Object Navigator中的Built-in Packages结点下。

表12-2 Oracle Developer Forms中的标准包

程 序 包	描 述
DDE	提供对Microsoft DDE(动态数据交换)的访问
DEBUG	提供在代码中查找问题的调试子程序
EXEC_SQL	提供在PL/SQL代码中使用的大多数DBMS_SQL包函数
FTREE	提供对分层树的操作
OLE2	提供对Microsoft OLE(对象链接和嵌入)版本2的访问
ORA_FFI	提供对外部函数的访问
ORA_NLS	提供关于语言环境的信息
ORA_PROF	提供描述代码的定时子程序
PECS	提供性能评价工具(Oracle公司不赞成使用)
STANDARD	提供PL/SQL内部过程和函数
STANDARD Extensions	提供Oracle内部过程和函数，用于扩展 PL/SQL
TEXT_IO	提供文件I/O
TOOL_ENV	提供对环境变量的访问
TOOL_ERR	提供错误处理子程序，包括 MESSAGE
TOOL_RES	提供对资源文件中字符串的访问
VBX	提供对处理VBX控制工具的访问
Web	提供用于显示Web文档的Show_Document子程序

Report Builder有许多与Forms相同的程序包，但是最有特点的是 SRW包。这个程序包中的组件提供了大多数对实现特定报表编程中需要的报表元素的访问。

Graphics有许多与Forms相同的程序包，另外附加了两个程序包： OG和TOOL_INT。OG包提供了数量庞大的组件，这些组件几乎可以完成对任意图表或显示元素的所有操作。TOOL_INT包提供了给其他产品传递参数的参数列表的建立工具。

China-pub.com
下载

12.4 检查库

前面有些部分讲述了PL/SQL库。库用来把PL/SQL程序聚集并存储到独立的存储模块中，以便使用起来更容易。通常，把PL/SQL代码作为数据库对象存储。由于PL/SQL作为客户端程序设计语言，所以找到一种途径把代码作为文件系统对象存储，使其能够连接到应用程序变得十分必要。

库的工作方式与所有的Oracle Developer产品一样。可以通过Oracle Developer任意一种产品或Procedure Builder建立作为独立模块的库。然后就可以把库连接到任何 Oracle Developer模块(表单、菜单、报表或图形显示)使模块中的所有组件对于任意对象可用。也可以把一个库连接到另一个模块。使用库中组件不需要加上库名作为前缀。

12.4.1 库的建立

在库中建立程序单元(过程、函数、程序包说明或程序包程序体)时，要在Object Navigator中找到库模块标题，并选择它。单击 Create工具建立一个新库，或者使用Open工具打开一个以前保存的库。选择库名下的Program Unit标题，单击Create工具建立一个程序单元。首先出现New Program Unit对话框，在这个对话框中可以选择要建立哪一种程序单元 (过程、函数、程序包说明或程序包程序体)，如图12-1所示。

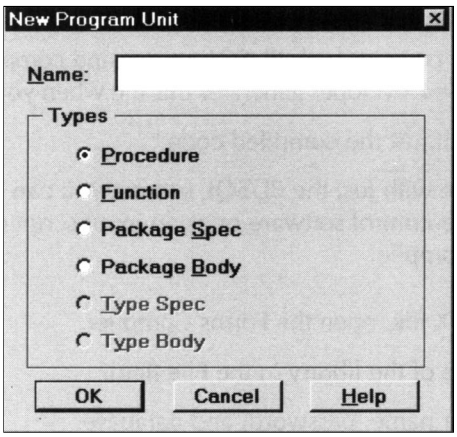

图12-1 New Program Unit对话框

选择之后单击OK按钮，出现如图12-2所示的包含选定程序单元类型模板的标准PL/SQL Editor。

代码编写结束之后，编译这个单元，通过 Save工具保存库模块。保存库时也设定库的名字，不能在 Navigator(导航器)中直接修改库名。确认已经对库中所有的程序单元进行了编译，如果不这样，使用库的模块会遇到问题。可以通过单元名上特殊符号存在或不存在来快速检查单元的状态：星号表示没有编译， (@)表示没有保存到磁盘。另外一个方法是在库名被选择时，选择 File|Compile All菜单项，来确保编译所有的单元或看到没有编译单元的错误。

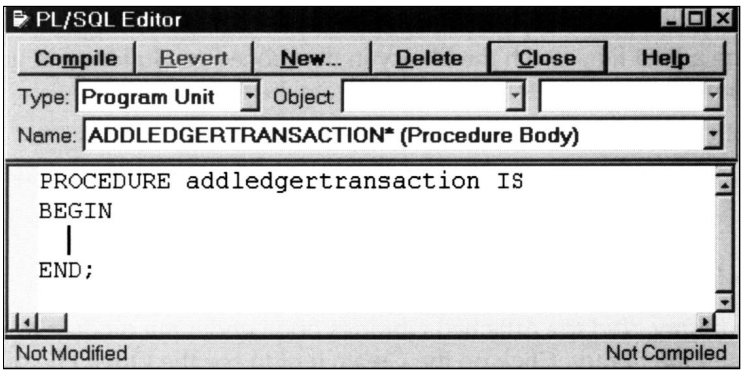

图12-2 PL/SQL Editor窗口

China-pub.com 下载

有三种类型的库文件：

- PLL：既包含PL/SQL源程序，也包含可以执行的已编译代码的文件， Oracle Developer在保存库时产生这种文件。
- PLX：只包含可以执行的已编译代码的文件。
- PLD：只包含PL/SQL源程序的文本文件，可以把这个文件作为对源控制软件的输入或能够包含和编译的PL/SQL输入脚本来使用。

1) 建立一个PLX文件，打开Forms Compiler(表单编译器)。

2) 在File域输入库的名字。

3) 输入用户名、口令和数据库。

4) 设置Module类型为FORM、MENU或LIBRARY域为LIBRARY。

5) 在“Write output to file”域输入要输出的PLX文件的文件名。图12-3举例说明了Talbot-Standard库的设置。

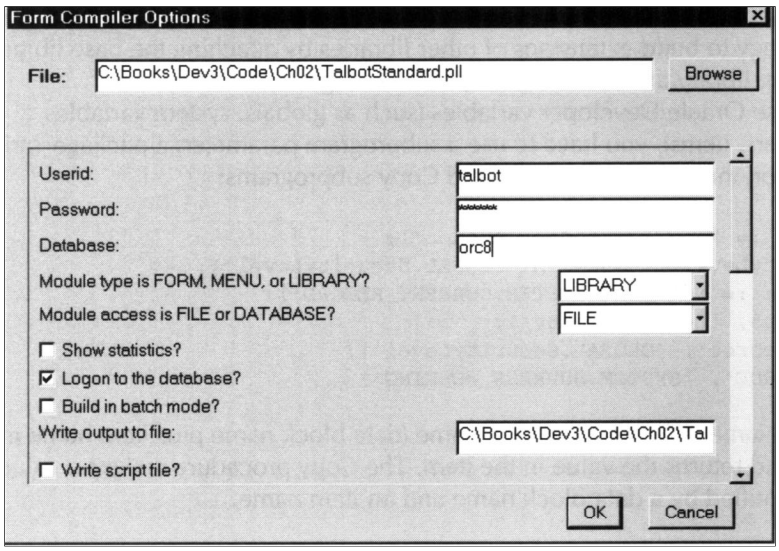

图12-3 为TalbotStandard库生成PLX文件

要建立PLD文件，在Procedure Builder中附加一个库，并在Interpreter(命令解释器)中使用EXPORT命令来生成库文本文件：

```
.EXPORT LIBRARY <library> FILE [<directory>]<name>.PLD
```

12.4.2 库的附加和使用

要附加一个库，在要附加库的模块下找到 Attached Library结点。单击Create工具，显示Open File对话框，选择要附加的库。

单击Attach按钮作为只读对象附加这个库。通过这个对象不能改变库中的任何内容。要想修改库的内容，在库模块标题下打开库进行修改。

提示 应该移除库文件名的任何路径信息。硬编码的路径意味着如果把它移到其他地方，必须改变并重新编译应用。反之，只需要改变注册变量FORMS60_PATH、REPORTS60_PATH或GRAPHICS60_PATH以包含库即可。

China-pub.com
下载

当附加一个库到另外一个模块时，模块就可以引用库的所有组件而不使用库名。在模块的Attached Library标题下库的顺序就是库名的搜索顺序。 Oracle Developer首先在Program Units部分查找程序单元。如果名字不存在，则按顺序查找每个附加库。因此可以通过把程序单元放到模块的Program Unit部分或在另外一个库前面放置一个超越库，实现特定程序单元的超越。可以使用这种方法来扩展或改编库，实现特定的用途，而不是重写或重新包装整个库。

附加一个库到另外一个库可以建立分层结构的库代码，使其他模块的使用只是简单的单击鼠标。也可以用这种方法通过附加基本库到其他库和重写程序单元的某些代码实现库的扩展。

使用Oracle Developer变量(例如全局、系统变量、参数、项)，通常要使用子程序参数、程序包变量或间接通过Name_In和Copy子程序。

```
vAmount := Name_In('Ledger.Amount');
vSecurityLevel := Name_In('GLOBAL.SecurityLevel');
vCurrent := Name_In('SYSTEM.CURRENT_RECORD');
Copy(4.05, 'Ledger.Amount');
Copy('Secret', 'GLOBAL.SecurityLevel');
Copy('TRUE', 'SYSTEM.SUPRESS_WORKING');
```

Name_In函数自变量为项名，返回值为项的值。 Copy过程把一个值连接到数据块名和项名指定的项。

注意 用子程序参数传递值比用Name_In或Copy引用变量要好。可以通过绑定变量语法(例如:Ledger.Amount)来完成。当使用Name_In/Copy方法时，把系统的两个部分通过对中间代码的直接连接联系起来，这不是一个好方法。使用子程序和程序包和参数使数据变得结构化、容易维护，库的重用性更好了。另外，如果为一个模块编写特定的代码并因此需要引用许多模块变量，应该把代码移到模块的Program Unit部分。这种方法能够直接访问变量。这些代码不能够重用，但由于设计问题，许多模块特定引用可能不能重用。

把本章学到的内容与第 11章介绍PL/SQL程序设计基础结合在一起，几乎覆盖了用 Oracle Developer开发非常复杂的应用程序的所有问题。由于许多事情都能在屏幕上做，所以并不太复杂，还在于为应用程序建立一组可重用系统组件从而建立有价值的应用程序的许多方法。通过使用PL/SQL的所有特性，同时可以确保应用能够完成用户想要做的事情，并且代码将来会工作得很好，增加效率，增加给予用户的价值。

下面的两章介绍需要确定的概念，以确保组件和应用程序做想要做的事：它们都是有关测试和调试的。