**PL/SQL块结构和组成元素**

**2.1   PL/SQL块**

**PL/SQL**程序由三个块组成，即声明部分、执行部分、异常处理部分。

**PL/SQL块的结构如下：**

**DECLARE    
    --声明部分: 在此声明PL/SQL用到的变量,类型及游标，以及局部的存储过程和函数  
  BEGIN  
    -- 执行部分:  过程及SQL 语句  , 即程序的主要部分  
  EXCEPTION  
    -- 执行异常部分: 错误处理  
  END;**

**其中：**执行部分不能省略。

**PL/SQL块可以分为三类：**

1.        **无名块或匿名块（anonymous）：**动态构造，只能执行一次，可调用其它程序，但不能被其它程序调用。

2.        **命名块（named）：**是带有名称的匿名块，这个名称就是标签。

3.        **子程序（subprogram）：**存储在数据库中的存储过程、函数等。当在数据库上建立好后可以在其它程序中调用它们。

4.        **触发器 （Trigger）：**当数据库发生操作时，会触发一些事件，从而自动执行相应的程序。

5.        **程序包（package）：**存储在数据库中的一组子程序、变量定义。在包中的子程序可以被其它程序包或子程序调用。但如果声明的是局部子程序，则只能在定义该局部子程序的块中调用该局部子程序。

**2.2   PL/SQL结构**

       PL/SQL块中可以包含子块；

       子块可以位于 PL/SQL中的任何部分；

       子块也即PL/SQL中的一条命令；

**2.3   标识符**

PL/SQL程序设计中的标识符定义与SQL 的标识符定义的要求相同。要求和限制有：

       标识符名不能超过30字符；

       第一个字符必须为字母；

       不分大小写；

       不能用’-‘(减号);

       不能是SQL保留字。

***提示:  一般不要把变量名声明与表中字段名完全一样,如果这样可能得到不正确的结果.***

**例如：**下面的例子将会删除所有的纪录，而不是’EricHu’的记录；

DECLARE  
   ename varchar2(**20**) :='EricHu';  
BEGIN  
  DELETE FROM scott.emp WHERE ename=ename;  
END;

    变量命名在PL/SQL中有特别的讲究，建议在系统的设计阶段就要求所有编程人员共同遵守一定的要求，使得整个系统的文档在规范上达到要求。下面是建议的命名方法：

|  |  |  |
| --- | --- | --- |
| **标识符** | **命名规则** | **例子** |
| 程序变量 | V\_name | V\_name |
| 程序常量 | C\_Name | C\_company\_name |
| 游标变量 | Cursor\_Name | Cursor\_Emp |
| 异常标识 | E\_name | E\_too\_many |
| 表类型 | Name\_table\_type | Emp\_record\_type |
| 表 | Name\_table | Emp |
| 记录类型 | Name\_record | Emp\_record |
| SQL\*Plus 替代变量 | P\_name | P\_sal |
| 绑定变量 | G\_name | G\_year\_sal |

**2.4   PL/SQL 变量类型**

在前面的介绍中，有系统的数据类型，也可以自定义数据类型。下表给出ORACLE类型和PL/SQL中的变量类型的合法使用列表：

**2.4.1  变量类型**

在ORACLE8i中可以使用的变量类型有：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 子类 | 说     明 | 范   围 | ORACLE限制 |
| CHAR | Character  String  Rowid  Nchar | 定长字符串      民族语言字符集 | 032767  可选,确省=1 | 2000 |
| VARCHAR2 | Varchar, String  NVARCHAR2 | 可变字符串  民族语言字符集 | 032767  4000 | 4000 |
| BINARY\_INTEGER |  | 带符号整数,为整数计算优化性能 |  |  |
| NUMBER(p,s) | Dec    Double precision  Integer  Int  Numeric  Real  Small int | 小数, NUMBER 的子类型  高精度实数  整数, NUMBER 的子类型  整数, NUMBER 的子类型  与NUMBER等价  与NUMBER等价  整数, 比 integer 小 |  |  |
| LONG |  | 变长字符串 | 0->2147483647 | 32,767字节 |
| DATE |  | 日期型 | 公元前4712年1月1日至公元后4712年12月31日 |  |
| BOOLEAN |  | 布尔型 | TRUE, FALSE,NULL | 不使用 |
| ROWID |  | 存放数据库行号 |  |  |
| UROWID |  | 通用行标识符，字符类型 |  |  |
|  |  |  |  |  |

**例1.**插入一条记录并显示；

DECLARE  
   Row\_id ROWID;  
   info    VARCHAR2(**40**);  
BEGIN  
  INSERT INTO scott.dept VALUES (**90**, '财务室', '海口')  
  RETURNING rowid, dname||':'||to\_char(deptno)||':'||loc  
  INTO row\_id, info;  
  DBMS\_OUTPUT.PUT\_LINE('ROWID:'||row\_id);  
  DBMS\_OUTPUT.PUT\_LINE(info);  
END;

其中：

RETURNING子句用于检索INSERT语句中所影响的数据行数，当INSERT语句使用VALUES 子句插入数据时，RETURNING 字句还可将列表达式、ROWID和REF值返回到输出变量中。在使用RETURNING 子句是应注意以下几点限制：

1．不能与DML语句和远程对象一起使用；

2．不能检索LONG 类型信息；

3．当通过视图向基表中插入数据时，**只能与单基表视图**一起使用。

**例2.**修改一条记录并显示:

DECLARE  
   Row\_id ROWID;  
   info   VARCHAR2(**40**);  
BEGIN  
  UPDATE dept SET deptno=**100** WHERE DNAME='财务室'  
  RETURNING rowid, dname||':'||to\_char(deptno)||':'||loc  
  INTO row\_id, info;  
  DBMS\_OUTPUT.PUT\_LINE('ROWID:'||row\_id);  
  DBMS\_OUTPUT.PUT\_LINE(info);  
END;

其中：

RETURNING子句用于检索被修改行的信息。当UPDATE语句修改单行数据时，RETURNING 子句可以检索被修改行的ROWID和REF值，以及行中被修改列的列表达式，并可将他们存储到PL/SQL变量或复合变量中；当UPDATE语句修改多行数据时，RETURNING 子句可以将被修改行的ROWID和REF值，以及列表达式值返回到复合变量数组中。在UPDATE中使用RETURNING 子句的限制与INSERT语句中对RETURNING子句的限制相同。

**例3.** 删除一条记录并显示

DECLARE  
   Row\_id ROWID;  
   info   VARCHAR2(**40**);  
BEGIN  
  DELETE dept WHERE DNAME='办公室'  
  RETURNING rowid, dname||':'||to\_char(deptno)||':'||loc  
  INTO row\_id, info;  
  DBMS\_OUTPUT.PUT\_LINE('ROWID:'||row\_id);  
  DBMS\_OUTPUT.PUT\_LINE(info);  
END;

其中：

RETURNING子句用于检索被删除行的信息：当DELETE语句删除单行数据时，RETURNING 子句可以检索被删除行的ROWID和REF值，以及被删除列的列表达式，并可将他们存储到PL/SQL变量或复合变量中；当DELETE语句删除多行数据时，RETURNING 子句可以将被删除行的ROWID和REF值，以及列表达式值返回到复合变量数组中。在DELETE中使用RETURNING 子句的限制与INSERT语句中对RETURNING子句的限制相同。

**2.4.2  复合类型**

    ORACLE 在 PL/SQL 中除了提供象前面介绍的各种类型外,还提供一种称为复合类型的类型---记录和表.

**2.4.2.1 记录类型**

记录类型类似于C语言中的结构数据类型，它把逻辑相关的、分离的、基本数据类型的变量组成一个整体存储起来，它必须包括至少一个标量型或RECORD 数据类型的成员，称作PL/SQL RECORD 的域(FIELD)，其作用是存放互不相同但逻辑相关的信息。在使用记录数据类型变量时，需要先在声明部分先定义记录的组成、记录的变量，然后在执行部分引用该记录变量本身或其中的成员。

**定义记录类型语法如下:**

**TYPE record\_name IS RECORD(  
   v1 data\_type1  [NOT NULL]  [:= default\_value ],  
   v2 data\_type2  [NOT NULL]  [:= default\_value ],  
   ......  
   vn data\_typen  [NOT NULL]  [:= default\_value ] );**

**例4 ：**

**DECLARE   
   TYPE test\_rec IS RECORD(  
         Name VARCHAR2(30) NOT NULL := '胡勇',  
         Info VARCHAR2(100));  
   rec\_book test\_rec;  
BEGIN  
   rec\_book.Name :='胡勇';  
   rec\_book.Info :='谈PL/SQL编程;';  
   DBMS\_OUTPUT.PUT\_LINE(rec\_book.Name||'  ' ||rec\_book.Info);  
END;**

可以用 SELECT语句对记录变量进行赋值,只要保证记录字段与查询结果列表中的字段相配即可。

**例5 ：**

DECLARE  
--定义与hr.employees表中的这几个列相同的记录数据类型  
   TYPE RECORD\_TYPE\_EMPLOYEES IS RECORD(  
        f\_name   hr.employees.first\_name%TYPE,  
        h\_date   hr.employees.hire\_date%TYPE,  
        j\_id     hr.employees.job\_id%TYPE);  
--声明一个该记录数据类型的记录变量  
   v\_emp\_record RECORD\_TYPE\_EMPLOYEES;  
  
BEGIN  
   SELECT first\_name, hire\_date, job\_id INTO v\_emp\_record  
   FROM employees  
   WHERE employee\_id = &emp\_id;  
  
   DBMS\_OUTPUT.PUT\_LINE('雇员名称：'||v\_emp\_record.f\_name  
             ||'  雇佣日期：'||v\_emp\_record.h\_date  
             ||'  岗位：'||v\_emp\_record.j\_id);  
END;

一个记录类型的变量只能保存从数据库中查询出的一行记录，若查询出了多行记录，就会出现错误。

**2.4.2.2 数组类型**

数据是具有相同数据类型的一组成员的集合。每个成员都有一个唯一的下标，它取决于成员在数组中的位置。在PL/SQL中，数组数据类型是VARRAY。

**定义VARRY数据类型语法如下：**

TYPE varray\_name IS VARRAY(size) OF element\_type [NOT NULL];

varray\_name是VARRAY数据类型的名称，size是下整数，表示可容纳的成员的最大数量，每个成员的数据类型是element\_type。默认成员可以取空值，否则需要使用NOT NULL加以限制。对于VARRAY数据类型来说，必须经过三个步骤，分别是：**定义、声明、初始化。**

**例6 ：**

DECLARE  
--定义一个最多保存5个VARCHAR(25)数据类型成员的VARRAY数据类型  
   TYPE reg\_varray\_type IS VARRAY(**5**) OF VARCHAR(**25**);  
--声明一个该VARRAY数据类型的变量  
   v\_reg\_varray REG\_VARRAY\_TYPE;  
  
BEGIN  
--用构造函数语法赋予初值  
   v\_reg\_varray := reg\_varray\_type  
         ('中国', '美国', '英国', '日本', '法国');  
  
   DBMS\_OUTPUT.PUT\_LINE('地区名称：'||v\_reg\_varray(**1**)||'、'  
                                    ||v\_reg\_varray(**2**)||'、'  
                                    ||v\_reg\_varray(**3**)||'、'  
                                    ||v\_reg\_varray(**4**));  
   DBMS\_OUTPUT.PUT\_LINE('赋予初值NULL的第5个成员的值：'||v\_reg\_varray(**5**));  
--用构造函数语法赋予初值后就可以这样对成员赋值  
   v\_reg\_varray(**5**) := '法国';  
   DBMS\_OUTPUT.PUT\_LINE('第5个成员的值：'||v\_reg\_varray(**5**));  
END;

**2.4.2.3 使用%TYPE**

定义一个变量，其数据类型与已经定义的某个数据变量(尤其是表的某一列)的数据类型相一致，这时可以使用%TYPE。

使用%TYPE特性的优点在于：

         所引用的数据库列的数据类型可以不必知道；

         所引用的数据库列的数据类型可以实时改变，容易保持一致，也不用修改PL/SQL程序。

**例7：**

**DECLARE  
   -- 用%TYPE 类型定义与表相配的字段  
   TYPE T\_Record IS RECORD(  
        T\_no emp.empno%TYPE,  
        T\_name emp.ename%TYPE,  
        T\_sal emp.sal%TYPE );  
   -- 声明接收数据的变量  
   v\_emp T\_Record;  
BEGIN  
   SELECT empno, ename, sal INTO v\_emp FROM emp WHERE empno=7788;  
   DBMS\_OUTPUT.PUT\_LINE  
    (TO\_CHAR(v\_emp.t\_no)||' '||v\_emp.t\_name||'  ' || TO\_CHAR(v\_emp.t\_sal));  
END;**

**例8：**

DECLARE  
   v\_empno emp.empno%TYPE :=&no;  
   Type t\_record is record (  
        v\_name   emp.ename%TYPE,  
        v\_sal    emp.sal%TYPE,  
        v\_date   emp.hiredate%TYPE);  
   Rec t\_record;  
BEGIN  
   SELECT ename, sal, hiredate INTO Rec FROM emp WHERE empno=v\_empno;  
   DBMS\_OUTPUT.PUT\_LINE(Rec.v\_name||'---'||Rec.v\_sal||'--'||Rec.v\_date);  
END;

2.4.3  使用%ROWTYPE

PL/SQL 提供%ROWTYPE操作符, 返回一个记录类型, 其数据类型和数据库表的数据结构相一致。

使用%ROWTYPE特性的优点在于：

         所引用的数据库中列的个数和数据类型可以不必知道；

         所引用的数据库中列的个数和数据类型可以实时改变，容易保持一致，也不用修改PL/SQL程序。

**例9：**

DECLARE  
    v\_empno emp.empno%TYPE :=&no;  
    rec emp%ROWTYPE;  
BEGIN  
    SELECT \* INTO rec FROM emp WHERE empno=v\_empno;  
    DBMS\_OUTPUT.PUT\_LINE('姓名:'||rec.ename||'工资:'||rec.sal||'工作时间:'||rec.hiredate);   
END;

2.4.4  LOB类型

    ORACLE提供了LOB (Large OBject)类型，用于存储大的数据对象的类型。ORACLE目前主要支持BFILE, BLOB, CLOB 及 NCLOB 类型。

**BFILE (Movie)**

    存放大的二进制数据对象，这些数据文件不放在数据库里，而是放在操作系统的某个目录里，数据库的表里只存放文件的目录。

**BLOB(Photo)**

    存储大的二进制数据类型。变量存储大的二进制对象的位置。大二进制对象的大小<=4GB。

**CLOB(Book)**

    存储大的字符数据类型。每个变量存储大字符对象的位置，该位置指到大字符数据块。大字符对象的大小<=4GB。

**NCLOB**

    存储大的NCHAR字符数据类型。每个变量存储大字符对象的位置，该位置指到大字符数据块。大字符对象的大小<=4GB。

**2.4.5  BIND 变量**

绑定变量是在主机环境中定义的变量。在PL/SQL 程序中可以使用绑定变量作为他们将要使用的其它变量。为了在PL/SQL 环境中声明绑定变量，使用命令VARIABLE。例如：

VARIABLE return\_code NUMBER  
VARIABLE return\_msg VARCHAR2(**20**)

可以通过SQL\*Plus命令中的PRINT 显示绑定变量的值。例如：

**PRINT return\_code  
PRINT return\_msg**

**例10：**

VARIABLE result NUMBER;  
BEGIN  
  SELECT (sal\***10**)+nvl(comm, **0**) INTO :result FROM emp   
  WHERE empno=**7844**;  
END;  
--然后再执行  
PRINT result

2.4.6  PL/SQL 表(TABLE)

定义记录表（或索引表）数据类型。它与记录类型相似，但它是对记录类型的扩展。它可以处理多行记录，类似于高级中的二维数组，使得可以在PL/SQL中模仿数据库中的表。

**定义记录表类型的语法如下：**

TYPE table\_name IS TABLE OF element\_type [NOT NULL]  
INDEX BY [BINARY\_INTEGER | PLS\_INTEGER | VARRAY2];

关键字INDEX BY表示创建一个主键索引，以便引用记录表变量中的特定行。

|  |  |
| --- | --- |
| 方法 | 描述 |
| EXISTS(n) | 如果集合的第n个成员存在，则返回true |
| COUNT | 返回已经分配了存储空间即赋值了的成员数量 |
| FIRST  LAST | FIRST：返回成员的最低下标值  LAST： 返回成员的最高下标值 |
| PRIOR(n) | 返回下标为n的成员的前一个成员的下标。如果没有则返回NULL |
| NEXT(N) | 返回下标为n的成员的后一个成员的下标。如果没有则返回NULL |
| TRIM | TRIM：删除末尾一个成员  TRIM(n) ：删除末尾n个成员 |
| DELETE | DELETE：删除所有成员  DELETE(n) ：删除第n个成员  DELETE(m, n) ：删除从n到m的成员 |
| EXTEND | EXTEND：添加一个null成员  EXTEND(n)：添加n个null成员  EXTEND(n,i)：添加n个成员，其值与第i个成员相同 |
| LIMIT | 返回在varray类型变量中出现的最高下标值 |

**例11：**

DECLARE  
  TYPE dept\_table\_type IS TABLE OF   
       dept%ROWTYPE INDEX BY BINARY\_INTEGER;  
  my\_dname\_table dept\_table\_type;  
  v\_count number(**2**) :=**4**;  
BEGIN  
  FOR int IN **1** .. v\_count LOOP  
    SELECT \* INTO my\_dname\_table(int) FROM dept WHERE deptno=int\***10**;  
  END LOOP;  
  FOR int IN my\_dname\_table.FIRST .. my\_dname\_table.LAST LOOP  
  DBMS\_OUTPUT.PUT\_LINE('Department number: '||my\_dname\_table(int).deptno);  
  DBMS\_OUTPUT.PUT\_LINE('Department name: '|| my\_dname\_table(int).dname);  
  END LOOP;  
END;

**例12：按一维数组使用记录表**

**DECLARE  
--定义记录表数据类型  
   TYPE reg\_table\_type IS TABLE OF varchar2(25)  
   INDEX BY BINARY\_INTEGER;  
--声明记录表数据类型的变量  
   v\_reg\_table REG\_TABLE\_TYPE;  
     
BEGIN  
   v\_reg\_table(1) := 'Europe';  
   v\_reg\_table(2) := 'Americas';  
   v\_reg\_table(3) := 'Asia';  
   v\_reg\_table(4) := 'Middle East and Africa';  
   v\_reg\_table(5) := 'NULL';  
  
   DBMS\_OUTPUT.PUT\_LINE('地区名称：'||v\_reg\_table (1)||'、'  
                                    ||v\_reg\_table (2)||'、'  
                                    ||v\_reg\_table (3)||'、'  
                                    ||v\_reg\_table (4));  
   DBMS\_OUTPUT.PUT\_LINE('第5个成员的值：'||v\_reg\_table(5));  
END;**

**例13：按二维数组使用记录表**

**DECLARE  
--定义记录表数据类型  
   TYPE emp\_table\_type IS TABLE OF employees%ROWTYPE  
   INDEX BY BINARY\_INTEGER;  
--声明记录表数据类型的变量  
   v\_emp\_table EMP\_TABLE\_TYPE;  
BEGIN  
   SELECT first\_name, hire\_date, job\_id INTO  
   v\_emp\_table(1).first\_name,v\_emp\_table(1).hire\_date, v\_emp\_table(1).job\_id  
   FROM employees WHERE employee\_id = 177;  
   SELECT first\_name, hire\_date, job\_id INTO  
   v\_emp\_table(2).first\_name,v\_emp\_table(2).hire\_date, v\_emp\_table(2).job\_id  
   FROM employees WHERE employee\_id = 178;  
  
   DBMS\_OUTPUT.PUT\_LINE('177雇员名称：'||v\_emp\_table(1).first\_name  
             ||'  雇佣日期：'||v\_emp\_table(1).hire\_date  
             ||'  岗位：'||v\_emp\_table(1).job\_id);  
   DBMS\_OUTPUT.PUT\_LINE('178雇员名称：'||v\_emp\_table(2).first\_name  
             ||'  雇佣日期：'||v\_emp\_table(2).hire\_date  
             ||'  岗位：'||v\_emp\_table(2).job\_id);  
END;**

2.5   运算符和表达式(数据定义)

**2.5.1  关系运算符**

|  |  |
| --- | --- |
| 运算符 | 意义 |
| = | 等于 |
| <> , != , ~= , ^= | 不等于 |
| < | 小于 |
| > | 大于 |
| <= | 小于或等于 |
| >= | 大于或等于 |

**2.5.2  一般运算符**

|  |  |
| --- | --- |
| 运算符 | 意义 |
| + | 加号 |
| - | 减号 |
| \* | 乘号 |
| / | 除号 |
| := | 赋值号 |
| => | 关系号 |
| .. | 范围运算符 |
| || | 字符连接符 |

**2.5.3  逻辑运算符**

|  |  |
| --- | --- |
| 运算符 | 意义 |
| IS NULL | 是空值 |
| BETWEEN  AND | 介于两者之间 |
| IN | 在一列值中间 |
| AND | 逻辑与 |
| OR | 逻辑或 |
| NOT | 取返,如IS NOT NULL, NOT IN |

**2.6   变量赋值**

在PL/SQL编程中，变量赋值是一个值得注意的地方，它的语法如下：

   variable  := expression ;

   variable 是一个PL/SQL变量, expression 是一个PL/SQL 表达式.

**2.6.1  字符及数字运算特点**

空值加数字仍是空值：NULL + < 数字> = NULL

空值加（连接）字符，结果为字符：NULL || <字符串> = < 字符串>

**2.6.2  BOOLEAN 赋值**

布尔值只有TRUE, FALSE及 NULL 三个值。如：

DECLARE  
  bDone BOOLEAN;  
BEGIN  
  bDone := FALSE;  
  WHILE NOT bDone LOOP  
  Null;  
  END LOOP;  
END;

**2.6.3  数据库赋值**

    数据库赋值是通过 SELECT语句来完成的，每次执行 SELECT语句就赋值一次，一般要求被赋值的变量与SELECT中的列名要一一对应。如：

**例14：**

**DECLARE  
  emp\_id    emp.empno%TYPE :=7788;  
  emp\_name  emp.ename%TYPE;  
  wages     emp.sal%TYPE;  
BEGIN  
  SELECT ename, NVL(sal,0) + NVL(comm,0) INTO emp\_name, wages   
  FROM emp WHERE empno = emp\_id;  
  DBMS\_OUTPUT.PUT\_LINE(emp\_name||'----'||to\_char(wages));  
END;**

***提示：不能将SELECT语句中的列赋值给布尔变量。***

**2.6.4  可转换的类型赋值**

       **CHAR 转换为 NUMBER：**

使用 TO\_NUMBER 函数来完成字符到数字的转换，如：

v\_total := TO\_NUMBER('100.0') + sal;

       **NUMBER 转换为CHAR：**

    使用 TO\_CHAR函数可以实现数字到字符的转换，如：

 v\_comm := TO\_CHAR('123.45') || '元' ;

       **字符转换为日期：**

使用 TO\_DATE函数可以实现  字符到日期的转换，如：

v\_date := TO\_DATE('2001.07.03','yyyy.mm.dd');

       **日期转换为字符**

使用 TO\_CHAR函数可以实现日期到字符的转换，如：

v\_to\_day := TO\_CHAR(SYSDATE, 'yyyy.mm.dd hh24:mi:ss') ;

**2.7   变量作用范围及可见性**

在PL/SQL编程中，如果在变量的定义上没有做到统一的话，可能会隐藏一些危险的错误，这样的原因主要是变量的作用范围所致。变量的作用域是指变量的有效作用范围，与其它高级语言类似，PL/SQL的变量作用范围特点是：

       变量的作用范围是在你所引用的程序单元（块、子程序、包）内。即从声明变量开始到该块的结束。

       一个变量（标识）只能在你所引用的块内是可见的。

       当一个变量超出了作用范围，PL/SQL引擎就释放用来存放该变量的空间（因为它可能不用了）。

       在子块中重新定义该变量后，它的作用仅在该块内。

**例15：**

DECLARE  
   Emess char(**80**);  
BEGIN  
  
   DECLARE  
      V1 NUMBER(**4**);  
   BEGIN  
      SELECT empno INTO v1 FROM emp WHERE LOWER(job)='president';  
      DBMS\_OUTPUT.PUT\_LINE(V1);  
   EXCEPTION  
      When TOO\_MANY\_ROWS THEN  
         DBMS\_OUTPUT.PUT\_LINE ('More than one president');  
   END;  
  
   DECLARE   
      V1 NUMBER(**4**);  
   BEGIN  
      SELECT empno INTO v1 FROM emp WHERE LOWER(job)='manager';  
   EXCEPTION  
      When TOO\_MANY\_ROWS THEN  
          DBMS\_OUTPUT.PUT\_LINE ('More than one manager');  
   END;  
  
EXCEPTION  
   When others THEN  
      Emess:=substr(SQLERRM,**1**,**80**);  
      DBMS\_OUTPUT.PUT\_LINE(emess);  
END;

**2.8   注释**

    在PL/SQL里，可以使用两种符号来写注释，即：

       使用双 ‘-‘ ( 减号) 加注释

PL/SQL允许用 – 来写注释，它的作用范围是只能在一行有效。如：

    V\_Sal  NUMBER(12,2); *-- 人员的工资变量。*

         使用 /\*   \*/  来加一行或多行注释，如：

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\* 文件名： department\_salary.sql      \*/*

*/\* 作 者： EricHu                     \*/*

*/\* 时 间： 2011-5-9                  \*/*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

***提示：***被解释后存放在数据库中的 PL/SQL 程序，一般系统自动将程序头部的注释去掉。只有在 PROCEDURE 之后的注释才被保留；另外程序中的空行也自动被去掉。

**2.9   简单例子**

**2.9.1   简单数据插入例子**

**例16：**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* 文件名： test.sql                  \*/  
/\* 说 明:  
       一个简单的插入测试，无实际应用。\*/  
/\* 作 者： EricHu                     \*/  
/\* 时 间： 2011-5-9                  \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
DECLARE  
  v\_ename   VARCHAR2(**20**) := 'Bill';  
  v\_sal       NUMBER(**7**,**2**) :=**1234.56**;  
  v\_deptno   NUMBER(**2**) := **10**;  
  v\_empno   NUMBER(**4**) := **8888**;  
BEGIN  
  INSERT INTO emp ( empno, ename, JOB, sal, deptno , hiredate )    
  VALUES (v\_empno, v\_ename, 'Manager', v\_sal, v\_deptno,   
            TO\_DATE('1954.06.09','yyyy.mm.dd') );  
  COMMIT;  
END;

2.9.2   简单数据删除例子

**例17：**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
/\* 文件名： test\_deletedata.sql      \*/  
/\* 说 明:  
       简单的删除例子，不是实际应用。 \*/  
/\* 作 者： EricHu                     \*/  
/\* 时 间： 2011-5-9                  \*/  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
DECLARE  
  v\_ename   VARCHAR2(**20**) := 'Bill';  
  v\_sal       NUMBER(**7**,**2**) :=**1234.56**;  
  v\_deptno   NUMBER(**2**) := **10**;  
  v\_empno   NUMBER(**4**) := **8888**;  
BEGIN  
  INSERT INTO emp ( empno, ename, JOB, sal, deptno , hiredate )    
VALUES ( v\_empno, v\_ename, ‘Manager’, v\_sal, v\_deptno,   
TO\_DATE(’**1954.06**.**09**’,’yyyy.mm.dd’) );  
  COMMIT;  
END;  
 DECLARE  
  v\_empno   number(**4**) := **8888**;  
BEGIN  
  DELETE FROM emp WHERE empno=v\_empno;  
  COMMIT;  
END;