

第9章 编程进阶——PL/SQL

本章通过实例的方式教会读者掌握 PL/SQL 的编程基本要素，从而完成复杂的管理任务。

9.1 节从实例出发，分析了完整的 PL/SQL 程序的结构。

9.2 节介绍 PL/SQL 程序的基本语法要素，包括常量、基本数据类型、复合数据类型、表达式和函数。

9.3 节介绍了条件控制和循环控制的流程控制结构。

9.4 节介绍了事务处理的 commit、rollback 和 savepoint 等 3 个命令的使用。

9.5 节介绍了如何定义、打开和提取游标的数据，如何使用主要的属性。

9.6 节介绍无参数过程和带参数过程的创建、查询、修改和使用方法。

9.7 节介绍了序列的创建和使用方法。

9.8 节介绍如何定义异常。

9.9 节通过一个综合实例介绍如何编写 PL/SQL 程序。

9.1 PL/SQL 程序结构

9.1.1 什么是 PL/SQL 程序

前面第 4 章学习的标准化的 SQL 语言对数据库进行各种操作，每次只能执行一条语句，语句以英文的分号“;”为结束标识，这样使用起来很不方便，同时效率较低，这是因为 Oracle 数据库系统不像 VB、VC 这样的程序设计语言，它侧重于后台数据库的管理，因此提供的编程能力较弱，而结构化编程语言对数据库的支持能力又较弱，如果一些稍微复杂点的管理任务都要借助编程语言来实现的话，这对管理员来讲是很大的负担。

正是在这种需求的驱使下，从 Oracle 6 开始，Oracle 公司在标准 SQL 语言的基础上发展了自己的 PL/SQL（Procedural Language/SQL，过程化 SQL 语言）语言，将变量、控制结构、过程和函数等结构化程序设计的要素引入了 SQL 语言中，这样就能够编制比较复杂的 SQL 程序了，利用 PL/SQL 语言编写的程序也称为 PL/SQL 程序块。

PL/SQL 程序块的主要特点如下。

- ☐ 具有模块化的结构。
- ☐ 使用过程化语言控制结构。
- ☐ 能够进行错误处理。



PL/SQL 程序块只能在【SQL Plus】、【SQLPlus Worksheet】等工具支持下以解释型方式执行，不能编译成可执行文件，脱离支撑环境执行。

9.1.2 PL/SQL 实例分析

下面将为前面建立的 tempuser 用户建立一个名为 testtable 的数据表。

在该表中有 recordnumber 整数字段和 currentdate 时间型字段，编制一个 PL/SQL 程序完成向该表中自动输入 100 个记录，要求 recordnumber 字段从 1 到 100，currentdate 字段为当前系统时间。

(1) 前面建立的 tempuser 用户默认的表空间为 USERS，因此，要想使用该用户能够使用表空间建立数据方案对象，必须首先给其赋予名为“RESOURCE”的角色。

(2) 以 system 用户、SYSDBA 身份登录数据库后，在【企业管理器】中按照修改用户的步骤进行操作，直到出现如图 9.1 所示的编辑用户的【角色】选项卡。

在【可用】下拉列表框中选择“RESOURCE”，单击  按钮将其添加到【已授予】列表框中。【默认值】单元格被选中，单击  按钮。

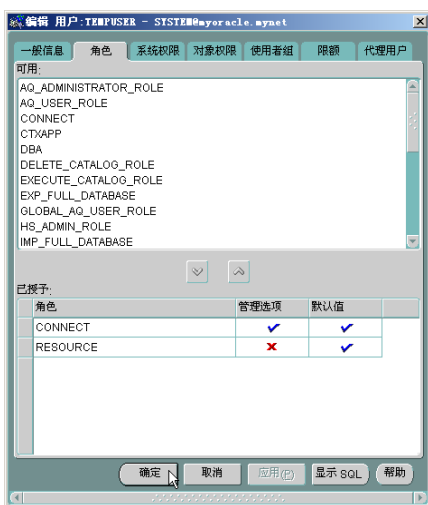


图 9.1 编辑用户的【角色】选项卡

(3) 读者也可以在【SQLPlus Worksheet】中直接执行如下 SQL 代码完成上述操作。

```
GRANT "RESOURCE" TO "TEMPUSER";
ALTER USER "TEMPUSER" DEFAULT ROLE ALL
```

【配套程序位置】：第 9 章\grantrole.sql。

(4) 按照创建数据表的操作步骤进行，直到出现如图 9.2 所示的创建表的【一般信息】选项卡。

在【名称】文本框中输入“testtable”。

在【方案】下拉列表框中选择“tempuser”。

在【表空间】下拉列表框中选择“users”。

选择【表】/【标准】单选钮。

选择【定义列】单选钮。

在【表列定义区】中输入两个数据列的定义。

完成设置后单击 **创建(C)** 按钮。

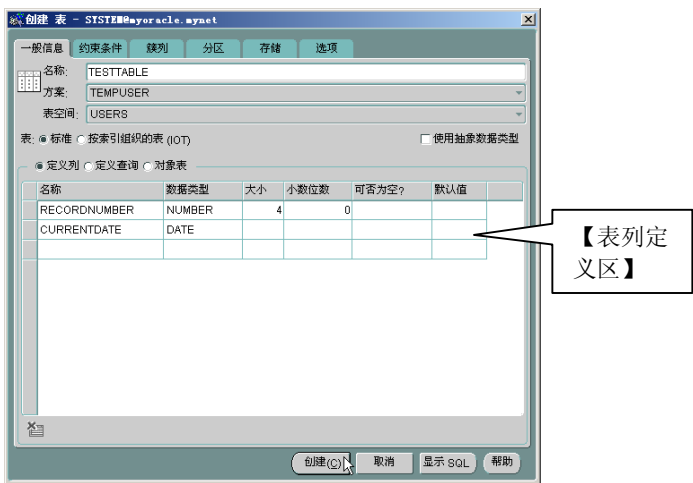


图 9.2 创建表的【一般信息】选项卡

(5) 读者也可以在【SQLPlus Worksheet】中直接执行如下 SQL 代码完成上述操作。

```
CREATE TABLE "TEMPUSER"."TESTTABLE" ("RECORDNUMBER" NUMBER(4) NOT
    NULL, "CURRENTDATE" DATE NOT NULL)
    TABLESPACE "USERS"
```

【配套程序位置】：第 9 章\createtesttable.sql。

(6) 以 tempuser 用户身份登录【SQLPlus Worksheet】，执行下列 SQL 代码完成向数据表 tempuser.testtable 中输入 100 个记录的功能。执行结果如图 9.3 所示。

```
set serveroutput on
declare
    maxrecords constant int:=100;
    i int :=1;
begin
    for i in 1..maxrecords loop
        insert into tempuser.testtable(recordnumber,currentdate)
            values(i,sysdate);
    end loop;
    dbms_output.put_line('成功录入数据!');
commit;
end;
```

【配套程序位置】：第 9 章\inserttesttable.sql。

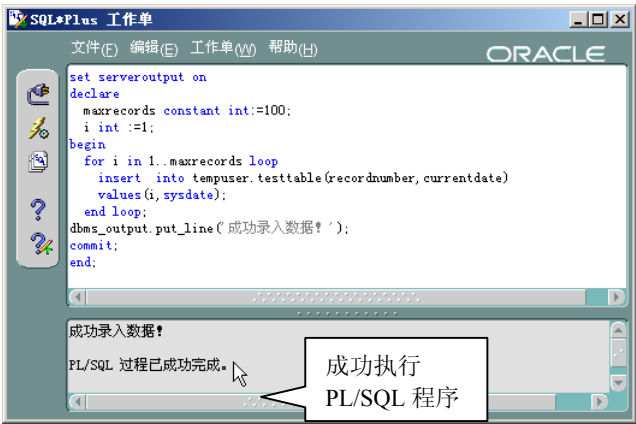


图 9.3 成功执行 PL/SQL 程序

(7) 在【SQLPlus Worksheet】中执行下列语句，查询插入的数据，结果如图 9.4 所示。

select * from tempuser.testtable;

【配套程序位置】：第 9 章\selecttesttable.sql。

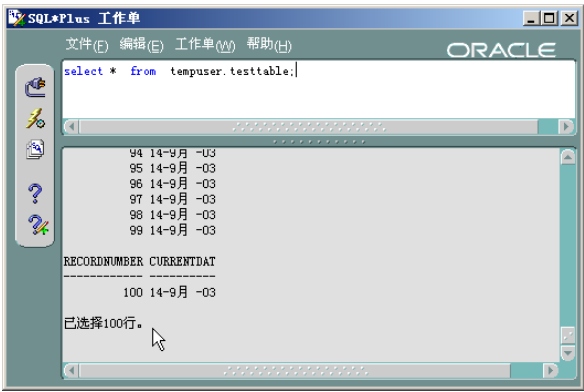


图 9.4 查询执行 PL/SQL 程序后的结果

对完成上述数据插入过程的 PL/SQL 程序的分析如表 9.1 所示。

表 9.1 PL/SQL 实例代码分析

程序代码	说明
set serveroutput on	允许服务器输出
declare	定义部分标识
maxrecords constant int:=100;	定义 maxrecords 为整型常量 100
i int :=1;	定义 i 为整型值变量，初值为 1
Begin	执行部分标识

续表

for i in 1..maxrecords loop	i 从 1 循环到 maxrecords
Insert into tempuser.testtable(recordnumber,currentdate) values (i,sysdate);	向数据表中插入数据
end loop;	结束循环
dbms_output.put_line('成功录入数据! ');	显示成功录入数据信息
commit;	提交结果
end;	结束执行



表中的 sysdate 为系统时间函数；dbms_output 为系统默认的程序包，put_line 为包中定义的方法，功能是输出信息；在 Oracle 中，所有对数据库数据的更改并没有直接操作数据库，而是放在叫工作区的内存里，只有在 commit 语句执行后，才发生永久更改。

9.1.3 PL/SQL 程序结构

结合上述实例进行分析，完整的 PL/SQL 程序结构可以分为 3 个部分。

1. 定义部分

以 Declare 为标识，在该部分中定义程序中要使用的常量、变量、游标和例外处理名称，PL/SQL 程序中使用的所有定义必须在该部分集中定义，而在高级语言里变量可以在程序执行过程中定义。

2. 执行部分

以 begin 为开始标识，以 end 为结束标识。该部分是每个 PL/SQL 程序所必备的，包含了对数据库的操作语句和各种流程控制语句。

3. 异常处理部分

该部分包含在执行部分里面，以 exception 为标识，对程序执行中产生的异常情况进行处理。一个完整的 PL/SQL 程序的总体结构如图 9.5 所示。

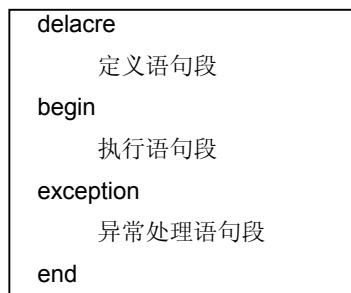


图 9.5 PL/SQL 程序的总体结构

有的程序比较简单，往往省略异常处理部分。下面开始介绍 PL/SQL 的一些基本语法要素。

9.2 基本语法要素

9.2.1 常量

1. 定义常量的语法格式

常量名 constant 类型标识符 [not null]:=值;



常量，包括后面的变量名都必须以字母开头，不能有空格，不能超过 30 个字符长度，同时不能和保留字同名，常（变）量名称不区分大小写，在字母后面可以带数字或特殊字符。括号内的 not null 为可选参数，若选用，表明该常（变）量不能为空值。

2. 实例

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序定义了名为 pi 的数字型常量，长度为 9。

执行结果如图 9.6 所示。

```
-----  
declare  
    pi constant number(9):=3.1415926;  
begin  
    commit;  
end;  
-----
```

【配套程序位置】：第 9 章\constantdefine.sql。

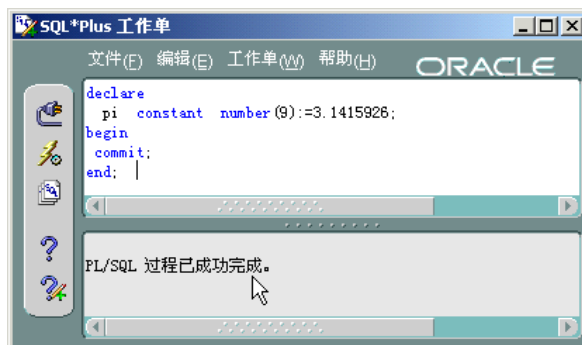


图 9.6 定义常量

9.2.2 基本数据类型变量

1. 基本数据类型

PL/SQL 中常用的基本数据类型如表 9.2 所示。

表 9.2 常见的数据基本类型

类型标识符	说明
Number	数字型
Int	整数型
Pls_integer	整数型，产生溢出时出现错误
Binary_integer	整数型，表示带符号的整数
Char	定长字符型，最大 255 个字符
Varchar2	变长字符型，最大 2000 个字符
Long	变长字符型，最长 2GB
Date	日期型
Boolean	布尔型（TRUE、FALSE、NULL 三者取一）



在 PL/SQL 中使用的数据类型和 Oracle 数据库中使用的数据类型，有的含义是完全一致的，有的是有不同的含义的。

2. 基本数据类型变量的定义方法

变量名 类型标识符 [not null]:=值;

3. 实例

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序定义了名为 age 的数字型变量，长度为 3，初始值为 26。执行结果如图 9.7 所示。

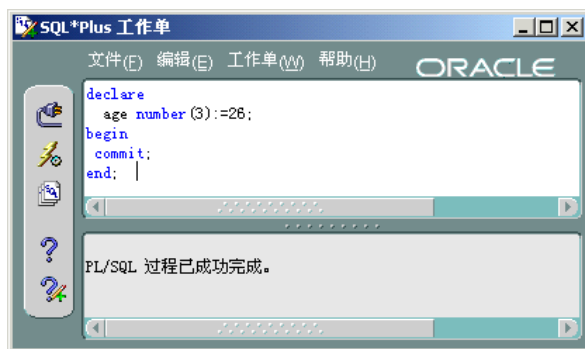


图 9.7 定义变量

```
declare
    age number(3):=26;
begin
    commit;
end;
```

【配套程序位置】：第 9 章\basicdatatypedefine.sql。

9.2.3 复合数据类型变量

下面介绍常见的几种复合数据类型变量的定义。

1. 使用%type 定义变量

为了让 PL/SQL 中变量的类型和数据表中的字段的数据类型一致，Oracle 9i 提供了%type 定义方法。这样当数据表的字段类型修改后，PL/SQL 程序中相应变量的类型也自动修改。

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序定义了名为 mydate 的变量，其类型和 tempuser.testtable 数据表中的 currentdate 字段类型是一致的。

执行结果如图 9.8 所示。

```
Declare
    mydate tempuser.testtable.currentdate%type;
begin
    commit;
end;
```

【配套程序位置】：第 9 章\typedefine.sql。

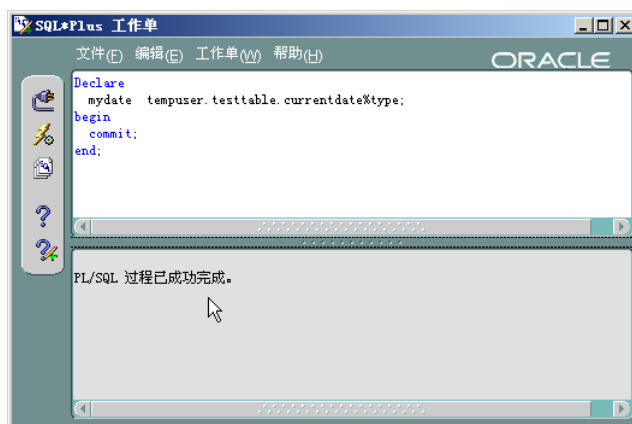


图 9.8 用%type 定义复合变量

2. 定义记录类型变量

很多结构化程序设计语言都提供了记录类型的数据类型，在 PL/SQL 中，也支持将多个基本数据类型捆绑在一起的记录数据类型。

下面的程序代码定义了名为 `myrecord` 的记录类型，该记录类型由整数型的 `myrecordnumber` 和日期型的 `mycurrentdate` 基本类型变量组成，`srecord` 是该类型的变量，引用记录型变量的方法是“记录变量名.基本类型变量名”。

程序的执行部分从 `tempuser.testtable` 数据表中提取 `recordnumber` 字段为 68 的记录的内容，存放在 `srecord` 复合变量里，然后输出 `srecord.mycurrentdate` 的值，实际上就是数据表中相应记录的 `currentdate` 的值。

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，执行结果如图 9.9 所示。

```
-----
set serveroutput on
declare
    type myrecord is record(
        myrecordnumber int,
        mycurrentdate date);
    srecord myrecord;
begin
    select * into srecord from tempuser.testtable where recordnumber=68;
    dbms_output.put_line(srecord.mycurrentdate);
end;
-----
```

【配套程序位置】：第9章\recordtypedefine.sql。



在 PL/SQL 程序中，`select` 语句总是和 `into` 配合使用，`into` 子句后面就是要被赋值的变量。



图 9.9 定义记录型复合变量

3. 使用%rowtype 定义变量

使用%type 可以使变量获得字段的数据类型，使用%rowtype 可以使变量获得整个记录的数据类型。比较两者定义的不同：变量名 数据表.列名%type，变量名 数据表%rowtype。

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序定义了名为 mytable 的复合类型变量，与 testtable 数据表结构相同，执行结果如图 9.10 所示。

```

-----
Declare
    mytable testtable%rowtype;
begin
    select * into mytable
    from tempuser.testtable
    where recordnumber=88;
    dbms_output.put_line(mytable.currentdate);
end;
-----

```

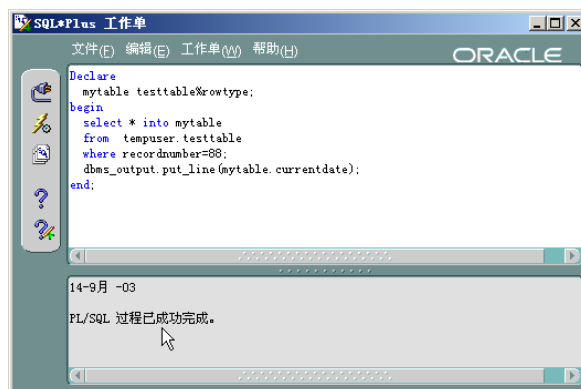


图 9.10 使用%rowtype 定义变量

【配套程序位置】：第 9 章\rowtypedefine.sql。

4. 定义一维表类型变量

表类型变量和数据表是有区别的，定义表类型变量的语法如下：

```

-----
type 表类型 is table of 类型 index by binary_integer;
表变量名 表类型;
-----

```

类型可以是前面的类型定义，index by binary_integer 子句代表以符号整数为索引，这样访问表类型变量中的数据方法就是“表变量名(索引符号整数)”。

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序定义了名为 tabletype1 和 tabletype2 的两个一维表类型，相当于一维数组。table1 和 table2 分别是两种表类型变量。

执行结果如图 9.11 所示。

```

-----
Declare
    type tabletype1 is table of varchar2(4) index by binary_integer;
    type tabletype2 is table of tempuser.testtable.recordnumber%type index by
binary_integer;
    table1 tabletype1;
    table2 tabletype2;
begin
    table1(1):='大学';
    table1(2):='大专';
    table2(1):=88;
    table2(2):=55;
    dbms_output.put_line(table1(1)||table2(1));
    dbms_output.put_line(table1(2)||table2(2));
end;
-----

```

【配套程序位置】：第9章\tabletypedefine1.sql。

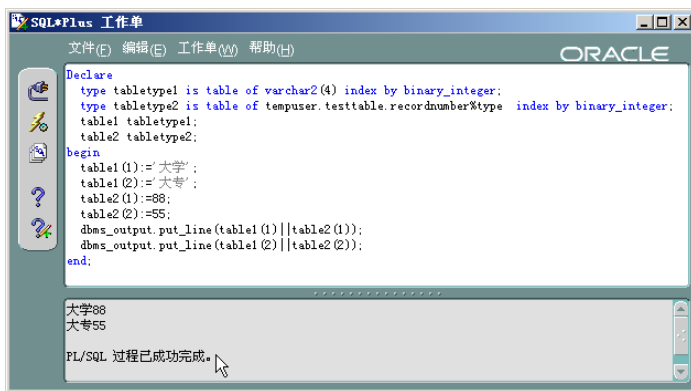


图 9.11 定义一维表类型变量



“||”是连接字符串的运算符。

5. 定义多维表类型变量

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序定义了名为 tabletype1 的多维表类型，相当于多维数组，table1 是多维表类型变量，将数据表 tempuser.testtable 中 recordnumber 为 60 的记录提取出来存放在 table1 中并显示。执行结果如图 9.12 所示。

```

-----
Declare

```

```

type tabletype1 is table of testtable%rowtype index by binary_integer;
table1 tabletype1;

begin
    select * into table1(60)
    from tempuser.testtable
    where recordnumber=60;
    dbms_output.put_line(table1(60).recordnumber||table1(60).currentdate);
end;

```

【配套程序位置】：第 9 章\tabletypedefine2.sql。



图 9.12 定义多维表类型变量



在定义好的表类型变量里，可以使用 count、delete、first、last、next、exists 和 prior 等属性进行操作，使用方法为“表变量名.属性”，返回的是数字。

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序定义了名为 tabletype1 的一维表类型，table1 是一维表类型变量，变量中插入 3 个数据，综合使用了表变量属性。执行结果如图 9.13 所示。

```

set serveroutput on

Declare
    type tabletype1 is table of varchar2(9) index by binary_integer;
    table1 tabletype1;

begin
    table1(1):='成都市';
    table1(2):='北京市';
    table1(3):='青岛市';
    dbms_output.put_line('总记录数: '||to_char(table1.count));
    dbms_output.put_line('第一条记录: '||table1.first);
    dbms_output.put_line('最后条记录: '||table1.last);

```

```

dbms_output.put_line(' 第二条的前一条记录: '||table1.prior(2));
dbms_output.put_line(' 第二条的后一条记录: '||table1.next(2));
end;

```

【配套程序位置】: 第9章\tabletypedefine3.sql。



图 9.13 使用表类型变量的属性

9.2.4 表达式

变量、常量经常需要组成各种表达式来进行运算，下面介绍在 PL/SQL 中常见表达式的运算规则。

1. 数值表达式

PL/SQL 程序中的数值表达式是由数值型常数、变量、函数和算术运算符组成的，可以使用的算术运算符包括+（加法）、-（减法）、*（乘法）、/（除法）和**（乘方）等。

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序定义了名为 result 的整数型变量，计算的是 $10+3*4-20+5**2$ 的值，理论结果应该是 27。执行结果如图 9.14 所示。

```

set serveroutput on
Declare
    result integer;
begin
    result:=10+3*4-20+5**2;
    dbms_output.put_line(' 运算结果是: '||to_char(result));
end;

```

【配套程序位置】: 第9章\datacompute.sql。

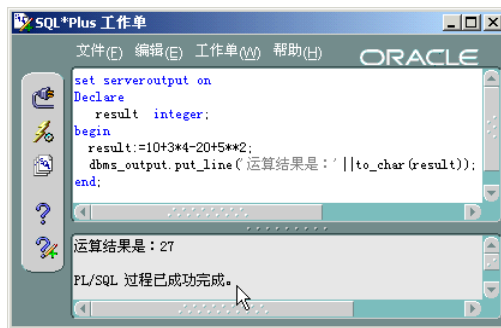


图 9.14 数值表达式运算



dbms_output.put_line 函数输出只能是字符串，因此利用 to_char 函数将数值型结果转换为字符型。

2. 字符表达式

字符表达式由字符型常数、变量、函数和字符运算符组成，唯一可以使用的字符运算符就是连接运算符“||”。

3. 关系表达式

关系表达式由字符表达式或数值表达式与关系运算符组成，可以使用的关系运算符包括以下 9 种。

- ☐ < 小于
- ☐ > 大于
- ☐ = 等于（不是赋值运算符:=）
- ☐ like 类似于
- ☐ in 在.....之中
- ☐ <= 小于等于
- ☐ >= 大于等于
- ☐ != 不等于
- ☐ between 在.....之间



关系型表达式运算符两边的表达式的数据类型必须一致。

4. 逻辑表达式

逻辑表达式由逻辑常数、变量、函数和逻辑运算符组成，常见的逻辑运算符包括以下 3 种。

- ☐ NOT: 逻辑非
- ☐ OR: 逻辑或
- ☐ AND: 逻辑与

运算的优先次序为 NOT、AND 和 OR。

9.2.5 函数

PL/SQL 程序中提供了很多函数供扩展功能，除了标准 SQL 语言的函数可以使用外，常见的数据类型转换函数有以下 3 个。

- ❑ To_char: 将其他类型数据转换为字符型。
- ❑ To_date: 将其他类型数据转换为日期型。
- ❑ To_number: 将其他类型数据转换为数值型。

以上介绍了 PL/SQL 中最基本的语法要素，下面介绍体现 PL/SQL 过程化编程思想的流程控制语句。

9.3 流程控制

PL/SQL 程序中的流程控制语句借鉴了许多高级语言的流程控制思想，但又有自己的特点。

9.3.1 条件控制

下面通过实例介绍条件控制语句的使用。

1. if..then..end if 条件控制

采用 if..then..end if 条件控制的语法结构如图 9.15 所示。

```
if 条件 then
    语句段;
end if;
```

图 9.15 if..then..end if 条件控制语法结构

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序判断两个整数变量的大小。执行结果如图 9.16 所示。

```
-----
set serveroutput on
declare
    number1 integer:=90;
    number2 integer:=60;
begin
    if number1>=number2 then
```

```

        dbms_output.put_line(' number1大于等于number2' );
    end if;
end;

```

【配套程序位置】：第 9 章\conditioncontrol1.sql。



图 9.16 if..then..end if 条件控制语句

2. if..then..else..end if 条件控制

采用 if..then..else..end if 条件控制的语法结构如图 9.17 所示。

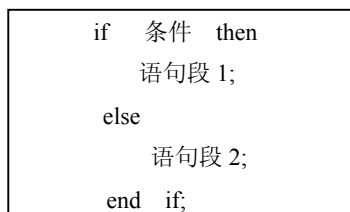


图 9.17 if..then..else..end if 条件控制语法结构

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序判断两个整数变量的大小，输出不同的结果。执行结果如图 9.18 所示。

```

set serveroutput on
declare
    number1 integer:=80;
    number2 integer:=90;
begin
    if number1>=number2 then
        dbms_output.put_line(' number1大于等于number2' );
    else
        dbms_output.put_line(' number1小于number2' );
    end if;
end;

```


【配套程序位置】：第9章\conditioncontrol2.sql。



图 9.18 if..then..else..end if 条件控制语句

3. if 嵌套条件控制

采用 if 嵌套条件控制的语法结构如图 9.19 所示。

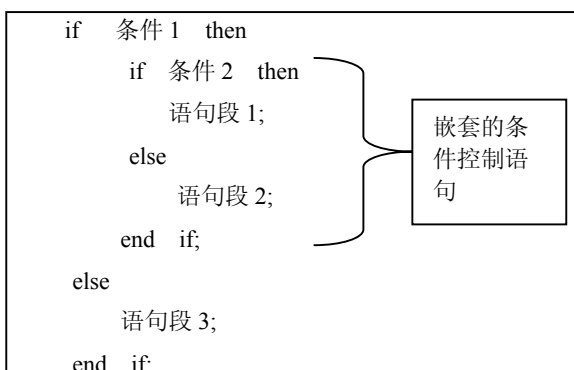


图 9.19 if 嵌套条件控制语法结构

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序判断两个整数变量的大小，输出不同的结果。

执行结果如图 9.20 所示。

```

set serveroutput on
declare
  number1 integer:=80;
  number2 integer:=90;
begin
  if number1<=number2 then
    if number1=number2 then
      dbms_output.put_line(' number1等于number2');
    end if;
  end if;
end;

```

```

else
    dbms_output.put_line(' number1小于number2');
end if;
else
    dbms_output.put_line(' number1大于number2');
end if;
end;

```

【配套程序位置】：第 9 章\conditioncontrol3.sql。



图 9.20 if 嵌套条件控制语句

9.3.2 循环控制

循环结构是按照一定逻辑条件执行一组命令，PL/SQL 中有 4 种基本循环结构，在它们基础上又可以演变出许多嵌套循环控制，这里介绍最基本的循环控制语句。

1. loop..exit..end loop 循环控制

采用 loop..exit..end loop 循环控制的语法结构如图 9.21 所示。

```

loop
    循环语句段;
    if 条件语句 then
        exit;
    else
        退出循环的处理语句段;
    end if;
end loop;

```

图 9.21 loop..exit..end loop 循环控制语法结构

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序将 number1 变量每次加 1，一直到等于 number2 为止，统计输出循环次数。

```

-----
set serveroutput on
declare
    number1 integer:=80;
    number2 integer:=90;
    i integer:=0;
begin
    loop
        number1:=number1+1;
        if number1=number2 then
            exit;
        else
            i:=i+1;
        end if;
    end loop;
    dbms_output.put_line('共循环次数:' || to_char(i));
end;
-----

```

执行结果如图 9.22 所示。



图 9.22 loop..exit..end loop 循环控制语句

【配套程序位置】：第 9 章\loopcontrol1.sql。

2. loop..exit..when..end loop 循环控制

采用 loop..exit..when..end loop 循环控制的语法结构与图 9.21 所示结构类似。

exit when 实际上就相当于

if 条件 then

exit;

end if;

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序将 number1 变量每次加 1，一直到等于 number2 为止，统计输出循环次数。

```
-----
set serveroutput on
declare
    number1 integer:=80;
    number2 integer:=90;
    i integer:=0;
begin
    loop
        number1:=number1+1;
        i:=i+1;
        exit when number1=number2;
    end loop;
    dbms_output.put_line('共循环次数:' || to_char(i));
end;
```

执行结果如图 9.23 所示。



图 9.23 loop..exit..when..end loop 循环控制语句

【配套程序位置】：第 9 章\loopcontrol2.sql。



when 循环控制结束条件比采用 if 的条件控制结束循环次数多 1 次。

3. while..loop..end loop 循环控制

采用 loop..exit..when..end loop 循环控制的语法如下。

while 条件 loop

 执行语句段;

end loop;

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序将 number1 变量每次加 1，一直到等于 number2 为止，统计输出循环次数。

```
-----
set serveroutput on
declare
    number1 integer:=80;
    number2 integer:=90;
    i integer:=0;
begin
    while number1<number2 loop
        number1:=number1+1;
        i:=i+1;
    end loop;
    dbms_output.put_line('共循环次数:' || to_char(i));
end;
-----
```

执行结果如图 9.24 所示。



图 9.24 while..loop..end loop 循环控制语句

【配套程序位置】：第9章\whilecontrol.sql。

4. for..in..loop..end 循环控制

采用 for..in..loop..end 循环控制的语法如下。

for 循环变量 in [reverse] 循环下界..循环上界 loop
循环处理语句段;

end loop;

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序,该程序通过循环变量 I 来控制 number1 增加次数,输出结果。执行结果如图 9.25 所示。

```
-----  
set serveroutput on  
declare  
    number1 integer:=80;  
    number2 integer:=90;  
    i integer:=0;  
begin  
    for i in 1..10 loop  
        number1:=number1+1;  
    end loop;  
    dbms_output.put_line('number1的值:' || to_char(number1));  
end;  
-----
```

【配套程序位置】: 第 9 章\forcontrol.sql。



图 9.25 for..in..loop..end 循环控制语句

9.4 事务处理

事务是 Oracle 9i 中进行数据库操作的基本单位,在 PL/SQL 程序中,可以使用 3 个事务处理控制命令。

9.4.1 commit 命令

commit 是事务提交命令。在 Oracle 9i 数据库中，为了保证数据的一致性，在内存中将为每个客户机建立工作区，客户机对数据库进行操作处理的事务都在工作区内完成，只有在输入 **commit** 命令后，工作区内的修改内容才写入到数据库上，称为物理写入，这样可以保证在任意的客户机没有物理提交修改以前，别的客户机读取的后台数据库中的数据是完整的、一致的，如图 9.26 所示。

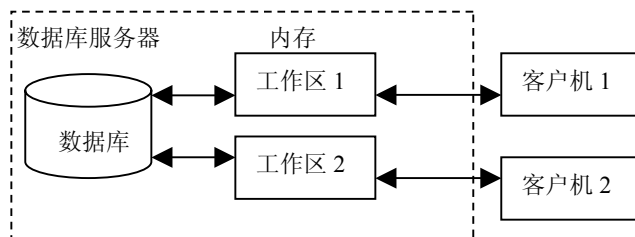


图 9.26 commit 命令示意图

在【SQLPlus Worksheet】中可以执行下列 PL/SQL 程序打开自动提交功能。这样每次执行 PL/SQL 程序都会自动进行事务提交。执行结果如图 9.27 所示。

```
set auto on;
```

【配套程序位置】：第 9 章\setautocommiton.sql。



图 9.27 打开自动提交功能

相应的，取消自动提交功能的 PL/SQL 程序如下。

```
set auto off;
```

【配套程序位置】：第 9 章\setautocommitoff.sql。

9.4.2 rollback 命令

rollback 是事务回滚命令，在尚未提交 **commit** 命令之前，如果发现 **delete**、**insert** 和 **update**

等操作需要恢复的话，可以使用 `rollback` 命令回滚到上次 `commit` 时的状态。

下面以 `delete` 命令为例演示如何回滚。

(1) 首先关闭自动提交功能。

(2) 在【SQLPlus Worksheet】中执行下列 SQL 程序，查看 `scott.emp` 数据表中的所有数据，执行结果如图 9.28 所示。

```
-----
select * from scott.emp;
-----
```

【配套程序位置】：第 9 章\selectemp.sql。



图 9.28 查看 `scott.emp` 表数据

(3) 在【SQLPlus Worksheet】中执行下列 SQL 程序，删除 `scott.emp` 数据表中的所有数据。执行结果如图 9.29 所示。

```
-----
delete from scott.emp;
-----
```

【配套程序位置】：第 9 章\deleteemp.sql。



图 9.29 删除 `scott.emp` 表数据

(4) 在【SQLPlus Worksheet】中执行下列 SQL 程序，查询删除数据表 `scott.emp` 后的结果。执行结果如图 9.30 所示。

```
-----
select * from scott.emp;
-----
```


【配套程序位置】：第9章\selectemp.sql。



图 9.30 查询删除 scott.emp 表数据后的结果

(5) 在【SQLPlus Worksheet】中执行以下 SQL 程序，完成事务的回滚。执行结果如图 9.31 所示。

 rollback;

【配套程序位置】：第9章\rollbackemp.sql。



图 9.31 执行 rollback 命令

(5) 在【SQLPlus Worksheet】中执行 selectemp.sql 文件，执行结果如图 9.28 所示，表明成功回滚了事务。

9.4.3 savepoint 命令

savepoint 是保存点命令。事务通常由数条命令组成，可以将每个事务划分成若干个部分进行保存，这样每次可以回滚每个保存点，而不必回滚整个事务。语法格式如下。

创建保存点：savepoint 保存点名；

回滚保存点：rollback to 保存点名；

下面介绍如何使用保存点。

(1) 在【SQLPlus Worksheet】中执行以下 SQL 程序，程序完成向 scott.emp 数据表中插入一条记录。执行结果如图 9.32 所示。

 insert into scott.emp(empno,ename,sal) values(9000,'wang',2500);

【配套程序位置】：第9章\insertemp.sql。



图 9.32 向 scott.emp 数据表中插入数据

(2) 在【SQLPlus Worksheet】中执行下列 SQL 代码，完成创建保存点的操作。执行结果如图 9.33 所示。

`savepoint insertpoint;`

【配套程序位置】: 第 9 章\createsavepoint.sql。

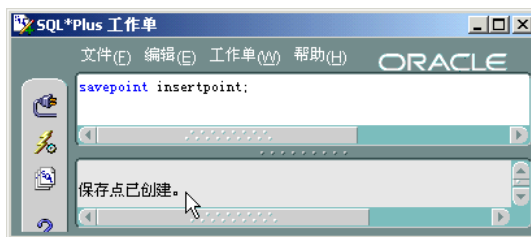


图 9.33 创建保存点

(3) 在【SQLPlus Worksheet】中可以执行其他的 SQL 语句。

(4) 在【SQLPlus Worksheet】中执行下列 SQL 代码，程序完成回滚到指定的保存点“insertpoint”。执行结果如图 9.34 所示。

`rollback to insertpoint;`

【配套程序位置】: 第 9 章\rollbacksavpoint.sql。

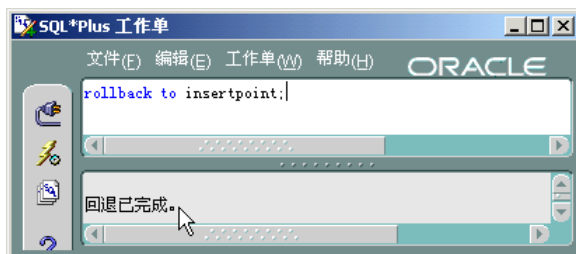


图 9.34 回滚保存点

(5) 可以执行 selectemp.sql 文件，查询回滚前后的数据。

9.5 游标

游标是从数据表中提取出来的数据，以临时表的形式存放在内存中，在游标中有一个数据指针，在初始状态下指向的是首记录，利用 `fetch` 语句可以移动该指针，从而对游标中的数据进行各种操作，然后将操作结果写回数据表中。

9.5.1 定义游标

游标作为一种数据类型，首先必须进行定义，其语法如下。

`cursor 游标名 is select 语句;`

`cursor` 是定义游标的关键词，`select` 是建立游标的数据表查询命令。

以 `scott` 用户连接数据库，在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序定义 `tempsal` 为与 `scott.emps` 数据表中的 `sal` 字段类型相同的变量，`mycursor` 为从 `scott.emp` 数据表中提取的 `sal` 大于 `tempsal` 的数据构成的游标。

执行结果如图 9.35 所示。

```
-----
set serveroutput on
declare
    tempsal scott.emp.sal%type;
    cursor mycursor is
        select * from scott.emp
        where sal>tempsal;
begin
    tempsal:=800;
    open mycursor;
end;
```

【配套程序位置】：第 9 章\cursordefine.sql。



图 9.35 定义游标

9.5.2 打开游标

要使用创建好的游标，接下来要打开游标，语法结构如下：

open 游标名;

打开游标的过程有以下两个步骤：

- (1) 将符合条件的记录送入内存。
- (2) 将指针指向第一条记录。

9.5.3 提取游标数据

要提取游标中的数据，使用 **fetch** 命令，语法形式如下。

fetch 游标名 **into** 变量名 1, 变量名 2,.....;

或

fetch 游标名 **into** 记录型变量名;

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序定义 **cursorrecord** 变量是游标 **mycursor** 的记录行变量，在游标 **mycursor** 的结果中找到 **sal** 字段大于 800 的第一个记录，显示 **deptno** 字段的内容。

执行结果如图 9.36 所示。

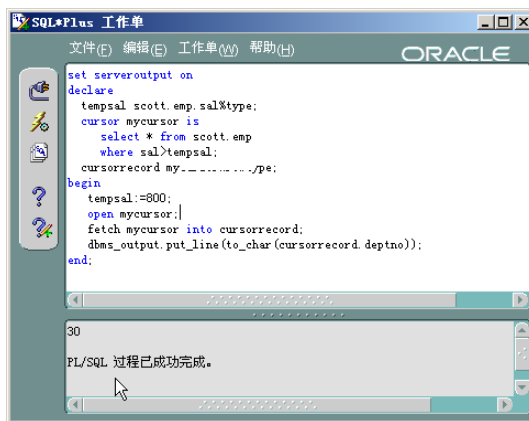


图 9.36 提取游标数据

```

set serveroutput on
declare
  tempsal scott.emp.sal%type;
  cursor mycursor is
    select * from scott.emp
    where sal>tempsal;
  cursorrecord mycursor%rowtype;
begin
  tempsal:=800;

```

```

open mycursor;
fetch mycursor into cursorrecord;
dbms_output.put_line(to_char(cursorrecord.deptno));
end;

```

【配套程序位置】: 第9章\cursorfetch.sql。

9.5.4 关闭游标

使用完游标后，要关闭游标，使用 close 命令，语法形式如下：

```
close 游标名;
```

9.5.5 游标的属性

游标提供的一些属性可以帮助编写 PL/SQL 程序，游标属性的使用方法为：游标名[属性]，例如 mycursor%isopen，主要的游标属性如下。

1. %isopen 属性

该属性功能是测试游标是否打开，如果没有打开游标就使用 fetch 语句将提示错误。

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序利用%isopen 属性判断游标是否打开。执行结果如图 9.37 所示。

```

-----
set serveroutput on
declare
    tempsal scott.emp.sal%type;
    cursor mycursor is
        select * from scott.emp
        where sal>tempsal;
    cursorrecord mycursor%rowtype;
begin
    tempsal:=800;
    if mycursor%isopen then
        fetch mycursor into cursorrecord;
        dbms_output.put_line(to_char(cursorrecord.deptno));
    else
        dbms_output.put_line('游标没有打开!');
    end if;
end;

```

【配套程序位置】: 第9章\isopenattribute.sql。



图 9.37 使用%isopen 属性

2. %found 属性

该属性功能是测试前一个 fetch 语句是否有值，有值将返回 true，否则为 false。

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序。该程序利用%found 属性判断游标是否有数据。

执行结果如图 9.38 所示。

```

-----
set serveroutput on
declare
  tempsal scott.emp.sal%type;
  cursor mycursor is
    select * from scott.emp
    where sal>tempsal;
  cursorrecord mycursor%rowtype;
begin
  tempsal:=800;
  open mycursor;
  fetch mycursor into cursorrecord;
  if mycursor%found then
    dbms_output.put_line(to_char(cursorrecord.deptno));
  else
    dbms_output.put_line('没有数据!');
  end if;
end;
-----

```

【配套程序位置】：第 9 章\ foundattribute.sql。



图 9.38 使用%found 属性

3. %notfound 属性

该属性是%found 属性的反逻辑，常被用于退出循环。

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序。该程序利用%notfound 属性判断游标是否没有数据。

执行结果如图 9.39 所示。

```

-----
set serveroutput on
declare
    tempsal scott.emp.sal%type;
    cursor mycursor is
        select * from scott.emp
        where sal>tempsal;
    cursorrecord mycursor%rowtype;
begin
    tempsal:=800;
    open mycursor;
    fetch mycursor into cursorrecord;
    if mycursor%notfound then
        dbms_output.put_line(to_char(cursorrecord.deptno));
    else
        dbms_output.put_line('发现数据!');
    end if;
end;
-----

```

【配套程序位置】：第9章\notfoundattribute.sql。



图 9.39 使用%notfound 属性

4. %rowcount 属性

该属性用于返回游标的数据行数。

在 SQLPlus Worksheet 的【代码编辑区】执行下列 PL/SQL 程序，该程序利用%rowcount 属性判断游标数据行数。

执行结果如图 9.40 所示。

```

-----
Set serveroutput on
declare
  tempsal scott.emp.sal%type;
  cursor mycursor is
    select * from scott.emp
    where sal>tempsal;
  cursorrecord mycursor%rowtype;
begin
  tempsal:=800;
  open mycursor;
  fetch mycursor into cursorrecord;
  dbms_output.put_line(to_char(mycursor%rowcount));
end;
-----

```

【配套程序位置】：第 9 章\rowcountattribute.sql。



若返回值为 0，表明游标已经打开，但没有提取出数据。



图 9.40 使用%rowcount 属性

9.6 过程

要想利用 PL/SQL 程序完成比较完整的数据库任务，需要进一步学习一些高级设计要素的内容。前面编写执行的 PL/SQL 程序，共同的特点是没有名称，只能存储为文件，然后通过执行文件的方式执行，因此称为无名块。与此对应的是在 PL/SQL 中也引入了高级程序设计的一些概念，其中最重要的就是过程。

过程就是高级程序设计语言中的模块的概念，将一些内部联系的命令组成一个个过程，通过参数在过程之间传递数据是模块化设计思想的重要内容。

9.6.1 创建过程

1. 过程的语法结构

完整的过程结构如下：

```
create or replace procedure 过程名 as
    声明语句段;
begin
    执行语句段;
exception
    异常处理语句段;
end;
```

2. 过程的特点

过程是有名称的程序块，as 关键词代替了无名块的 declare。

3. 创建过程实例

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序将创建名为 tempprocedure 的过程，create 是创建过程的标识符，replace 表示若同名过程存在将覆盖原过程。该过程定义了一个变量，其类型和 testtable 数据表中的 currentdate 字段类型相同，都是日期型，将数据表中的 recordnumber 字段为 88 的 currentdate 字段内容送入变量中，然后输出结果。

```
-----  
set serveroutput on  
create or replace procedure tempuser.tempprocedure as  
    tempdate    tempuser.testtable.currentdate%type;  
begin  
    select  currentdate  
    into    tempdate  
    from    testtable  
    where   recordnumber=88;  
    dbms_output.put_line(to_char(tempdate));  
end;  
-----
```

【配套程序位置】：第 9 章\createprocedure.sql。

执行结果如图 9.41 所示。



图 9.41 创建过程



创建过程并不会直接输出结果，只是和创建其他数据库对象一样，是一个数据定义命令。

9.6.2 查询过程

登录【企业管理器】，在【管理目标导航树】里选择【网络】/【数据库】/【myoracle.mynet】/【方案】/【过程】/【TEMPUSER】选项，出现如图 9.42 所示的创建好的过程。

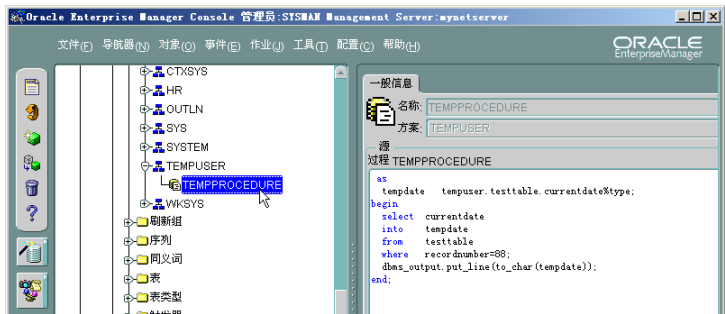


图 9.42 创建好的过程

9.6.3 修改过程

- (1) 在【SQLPlus Worksheet】的【菜单栏】选择【文件】/【打开】菜单命令，将创建过程的 createprocedure.sql 文件调出进行修改，修改完毕后重新执行创建过程。
- (2) 在【企业管理器】里选中要修改的过程，用鼠标右键单击，在出现的快捷菜单里选择【查看/编辑详细资料】选项，如图 9.43 所示。
- (3) 出现如图 9.44 所示的编辑过程的【一般信息】选项卡。在【文本编辑区】可以编辑该过程，单击 **确定** 按钮将更新该过程，单击 **编译** 按钮将编译该过程的 PL/SQL 源代码，使该过程可以在数据库中存储和执行。

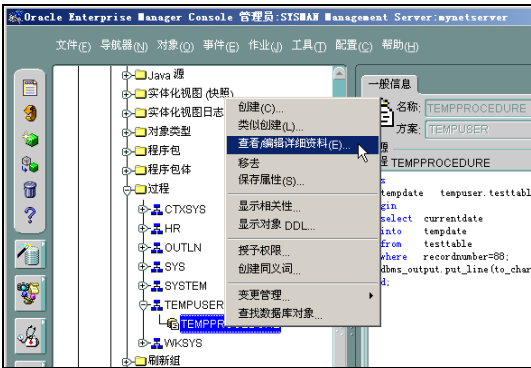


图 9.43 选择编辑过程

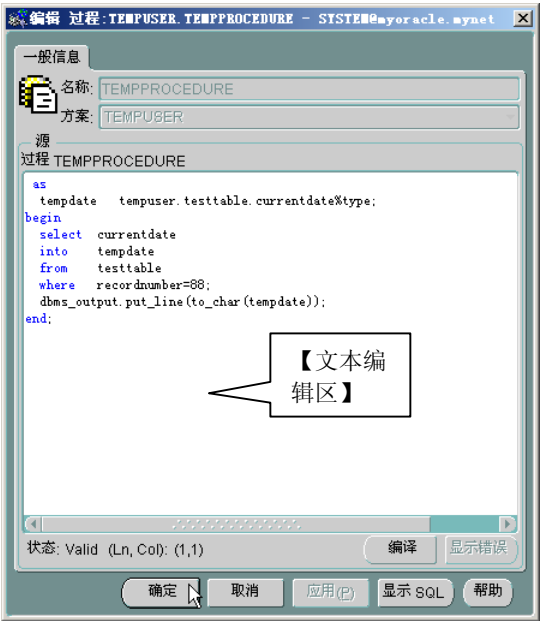


图 9.44 编辑过程的【一般信息】选项卡

9.6.4 执行过程

要执行创建的过程，必须通过主程序来调用过程。

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，执行结果如图 9.45 所示。

```
set serveroutput on  
begin  
    tempprocedure;  
end;
```

【配套程序位置】：第 9 章\executeprocedure.sql。

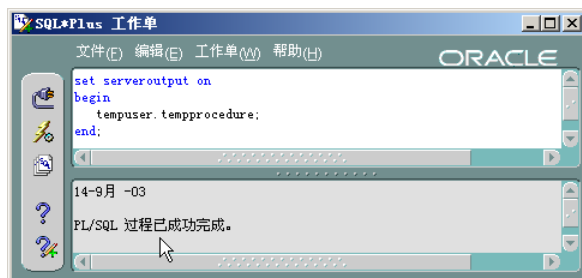


图 9.45 过程执行结果



在 Oracle 中，创建好的过程可以被任何程序调用。

9.6.5 带参数的过程

前面介绍的过程没有参数，主程序和过程没有数据的传递，下面介绍带参数的过程的设计和使用。

1. 参数类型

在 PL/SQL 过程中，可以有 3 种类型的参数。

- ☐ in 参数：读入参数，主程序向过程传递参数值。
- ☐ out 参数：读出参数，过程向主程序传递参数值。
- ☐ in out 参数：双向参数，过程与主程序双向交流数据。

2. 定义带参数的过程

在下面的 PL/SQL 程序代码中，将创建三个调用参数。

- ☐ tempdeptno: 类型为 in，与 scott.dept.deptno 的类型一致，为数值型。
- ☐ tempdname: 类型为 out，与 scott.dept.dname 的类型一致，为字符型。
- ☐ temploc: 类型为 in out，与 scott.dept.loc 类型一致，为字符型。

创建两个过程内参数。

- ☐ loc1: 与 scott.dept.loc 的类型一致，为字符型。
- ☐ dname1: 与 scott.dept.dname 的类型一致，为字符型。

该带参数的过程的功能是从数据表 scott.dept 中寻找 deptno 字段等于 tempdeptno 调用参

数值的 `dname` 和 `loc` 字段，和其他字符组合，送给两个出口参数。

以 `system` 用户名、`sysdba` 身份登录【SQLPlus Worksheet】，执行下列 PL/SQL 程序，执行结果如图 9.46 所示。

```
-----
Set serveroutput on
create or replace procedure scott.tempprocedure(
    tempdeptno in scott.dept.deptno%type,
    tempdname out scott.dept.dname%type,
    temploc in out scott.dept.loc%type)as
    loc1 scott.dept.loc%type;
    dname1 scott.dept.dname%type;
begin
    select loc into loc1
    from scott.dept
    where deptno=tempdeptno;
    select dname into dname1
    from scott.dept
    where deptno=tempdeptno;
    temploc:='地址:' || loc1;
    tempdname:='姓名' || dname1;
end;
-----
```

【配套程序位置】：第9章\createscottprocedure.sql。



图 9.46 创建带参数的过程



调用参数分割用“,”号。

3. 使用带参数的过程

在主程序中的实际参数和过程中的形式参数的传递有很多种办法，这里推荐读者采用一对应的办法，按对应的位置传递参数。要求实际参数和形式参数在数据类型和位置排列上做到完全一致。

在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，该程序调用带参数的过程 scott.tempprocedure，实际参数为（10，”，”）。

执行结果如图 9.47 所示。

```
-----  
set serveroutput on  
declare  
    myno scott.dept.deptno%type;  
    mydname scott.dept.dname%type;  
    myloc scott.dept.loc%type;  
begin  
    myno:=10;  
    mydname:='';  
    myloc:='';  
    scott.tempprocedure(myno,mydname,myloc);  
    dbms_output.put_line(myno);  
    dbms_output.put_line(mydname);  
    dbms_output.put_line(myloc);  
end;
```

读者可以尝试改变这 3 个参数，测试带参数过程

【配套程序位置】：第 9 章\executescottprocedure.sql。

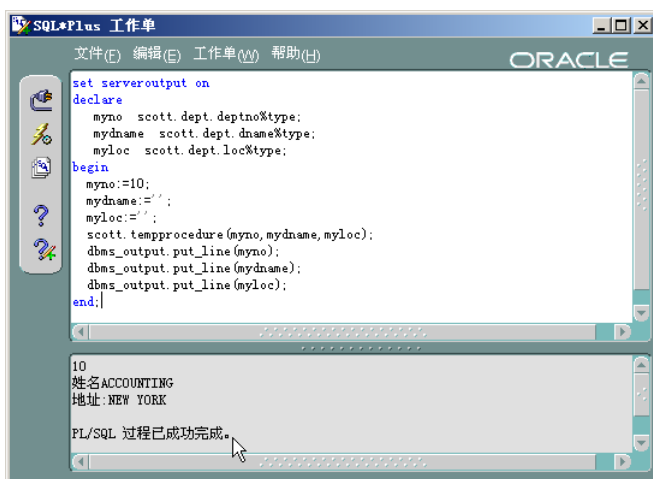


图 9.47 使用带参数过程

读者可以尝试改变参数值，然后测试过程执行结果。

9.7 序列

序列是 Oracle 9i 提供的用于按照设定的规则自动产生数据的方案对象。在某些数据表的结构中，有些字段需要这种特性。比如，对于某个学生数据表的学号关键字段，用户可以希望在录入数据时，能够自动在上一个记录的学号字段上自动加 1 等。由于 Oracle 9i 提供的 16 种基本数据类型并没有这样的功能，可以通过序列方案对象来实现。

9.7.1 序列的创建

下面介绍在【企业管理器】中如何创建序列。

(1) 在【企业管理器】中选择【myoracle.mynet】/【方案】/【序列】选项，单击鼠标右键，在出现的快捷菜单里选择【创建】选项，如图 9.48 所示。

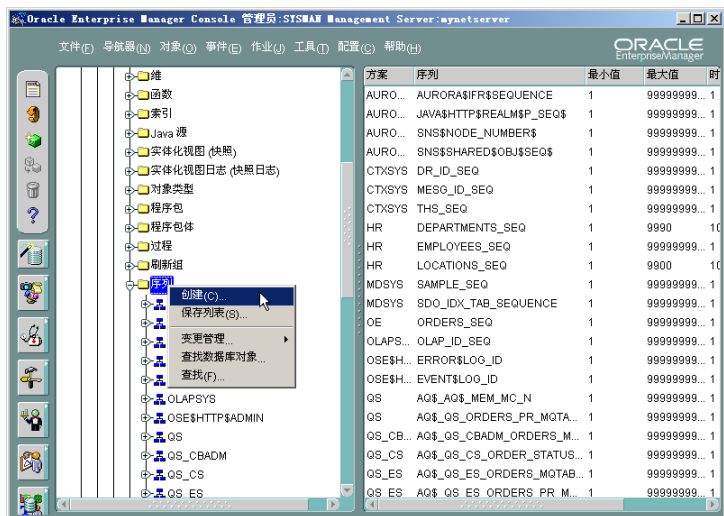


图 9.48 选择创建序列

(2) 出现如图 9.49 所示的创建序列的【一般信息】选项卡。

在【名称】文本框里输入待定义的序列的名称“TEMPSEQUENCE”。

在【方案】下拉列表框里选择序列所属的用户名“SCOTT”。

序列【类型】参数有两个选项。若选择【升序】单选钮，则表示将创建从初始值向最大值递增的序列，这是创建序列时的默认设置；若选择【降序】单选钮，则表示将创建从初始值向最小值递减的序列。

对【值】可以进行设置的参数如下。

□ 在【最小值】文本框里设置序列允许的最小值。创建序列时该字段最初为空。如果单击【创建】按钮时该字段为空，则对升序序列使用默认值 1，而对降序序列使用默认值 -1026。

□ 在【最大值】文本框里设置序列允许的最大值。创建序列时该字段最初为空。如果单击【创建】按钮后该字段为空，则将对升序序列使用默认值 1027，而对降序序列使用默认

值-1。

❑ 在【时间间隔】文本框里设置递增序列递增的间隔数值（升序序列）或递减序列递减的间隔数值（降序序列）。创建序列时该字段最初为空，如果单击【创建】按钮后该字段为空，将使用默认值 1，该字段只能为正整数。


❑ 在【初始值】文本框里设置序列的起始值。如果单击【创建】按钮后该字段为空，对升序序列将使用该序列默认的最小值，对降序序列将使用该序列默认的最大值。

对【选项】可以设置的参数如下。

❑ 若选择【循环值】复选框，则表示指定在达到序列最小值或最大值之后，序列应继续生成值。对升序序列来说，在达到最大值后将生成最小值。对降序序列来说，在达到最小值后将生成最大值。如果未选择该复选框，序列将在达到最小值或最大值后停止生成任何值。默认情况下是未选择状态。

❑ 若选择【排序值】复选框，则指定序列号要按请求次序生成，默认情况下是未选择状态。

❑ 在【高速缓存】中设置由数据库预分配并存储的值的数目参数。若选择【默认值】单选按钮，则表示将设置默认值为 20，默认情况下选择此选项；若选择【无高速缓存】单选按钮，则表示指定不预分配序列值；若选择【大小】单选按钮，则表示在文本框里输入可接受的值，最小值为 2，对循环序列来说，该值必须小于循环中值的个数。如果序列能够生成的值数的上限小于高速缓存大小，则高速缓存大小将自动改换为该上限数。

完成设置后单击  按钮。


(3) 成功创建序列后，出现如图 9.50 所示界面。单击  按钮。



图 9.49 创建序列的【一般信息】选项卡 图 9.50 【成功创建序列】界面

(3) 读者也可以在【SQLPlus Worksheet】中执行下列 SQL 程序创建序列。

```
CREATE SEQUENCE "SCOTT"."TEMPSEQUENCE"
  INCREMENT BY 1 START WITH 1
  MAXVALUE 1.0E28 MINVALUE 1
  NOCYCLE CACHE 20 NOORDER
```


【配套程序位置】：第9章\createsequence.sql。

9.7.2 序列的使用

下面介绍在向数据表中插入数据时如何使用序列。

(1) 首先为实例建立一个数据表“SCOTT.SEQUENCE_TABLE”，为简化起见，该数据表仅包含一个类型为“NUMBER”的数据列“NO”。

在如图9.51所示的创建表的【一般信息】选项卡中进行如下设置。

在【名称】文本框中输入“SEQUENCE_TABLE”。

在【方案】下拉列表框中选择“SCOTT”。

在【表空间】下拉列表框中选择“USERS”。

在【名称】单元格中输入“NO”，在【数据类型】下拉列表框单元格中选择“NUMBER”。完成设置后单击“创建(C)”按钮。

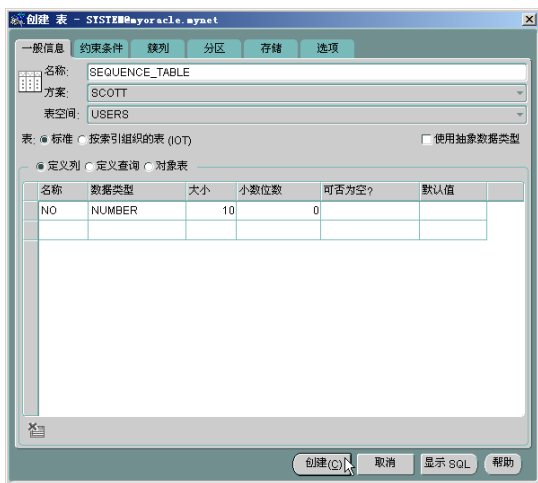


图 9.51 创建表的【一般信息】选项卡

(2) 读者也可以在【SQLPlus Worksheet】中执行下列 SQL 代码创建数据表“SCOTT.SEQUENCE_TABLE”。

```
CREATE TABLE "SCOTT"."SEQUENCE_TABLE" ("NO" NUMBER(10) NOT NULL)
TABLESPACE "USERS"
```

【配套程序位置】：第9章\createsequencetable.sql。

(3) 在插入新的记录时，使用刚创建的“TEMPSEQUENCE”序列来自动产生“NO”数据列的值。在【SQLPlus Worksheet】里执行下面的 SQL 代码，执行的结果如图9.52所示。

```
INSERT INTO SCOTT.SEQUENCE_TABLE(NO)
VALUES (SCOTT.TEMPSEQUENCE.NEXTVAL);
```

【配套程序位置】：第 9 章\insertsequencetable.sql。

“SCOTT.TEMPSEQUENCE.NEXTVAL” 表分配下一个惟一的、可用的序列号。

执行“SCOTT.TEMPSEQUENCE.NEXTVAL”后，可以使用“SCOTT.TEMPSEQUENCE.CURRVAL”来标识上一个已经存储的序列值。

(4) 在【SQLPlus Worksheet】中可以执行查询数据表“SCOTT.SEQUENCE_TABLE”数据的语句。执行结果如图 9.53 所示，表明序列“SCOTT.SEQUENCE”产生的值已经成功录入数据表中。

```
select * from scott.sequence_table;
```

【配套程序位置】：第 9 章\selectsequencetable.sql。

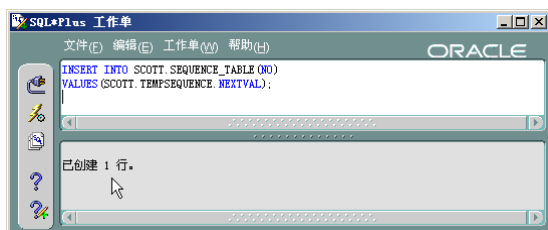


图 9.52 利用序列产生数据

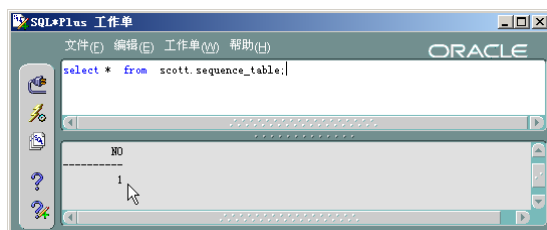


图 9.53 查询序列数据的结果

9.7.3 序列的删除

下面介绍在【企业管理器】中如何删除序列。

(1) 在创建好的序列“SEQUENCE1”上用单击鼠标右键，在出现的快捷菜单里选择【移去】选项，如图 9.54 所示。

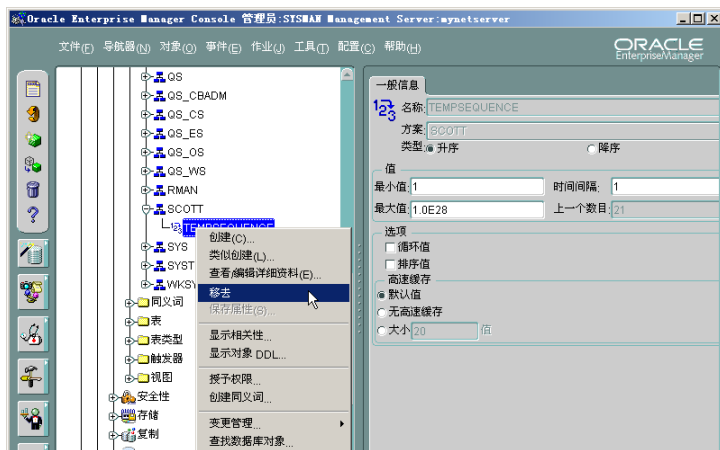


图 9.54 选择删除序列


(2) 出现如图 9.55 所示的【删除序列确认】界面，单击  按钮。



图 9.55 【删除序列确认】界面

9.8 异常处理

在设计 PL/SQL 程序时，经常会发生这样或那样的错误，异常处理就是针对错误进行处理的程序段，Oracle 9i 中的异常处理分为系统预定义异常处理和自定义异常处理两部分。

9.8.1 系统预定义异常处理

系统预定义异常处理是针对 PL/SQL 程序编译、执行过程中发生的问题进行处理的程序。下列代码为正确代码，在【SQLPlus Worksheet】中能够顺利执行。

```
-----
set serveroutput on
declare
    tempno integer:=90;
begin
    tempno:=tempno+1;
end;
-----
```

【配套程序位置】：第 9 章\correctplsql.sql。

下列代码为错误代码，在【SQLPlus Worksheet】中的执行结果如图 9.56 所示。

```
-----
set serveroutput on
declare
    tempno integer:=100;
begin
    tempno=tempno+1;
end;
-----
```

正确代码应该是：tempno:=tempno+1;

【配套程序位置】：第 9 章\wrongplsql.sql。

由于代码有错误，因此将激活系统预定义的异常处理，并得出如下提示信息。



图 9.56 调用系统异常处理

Oracle 9i 提供了很多异常处理，读者可以尝试修改可以正常运行的程序，并执行修改后的程序，就可以发现调用了哪些异常处理，下面着重介绍如何自定义异常处理。

9.8.2 自定义异常处理

1. 定义异常处理

定义异常处理的语法如下：

```
declare
    异常名 exception;
```

2. 触发异常处理

触发异常处理的语法如下：

```
raise 异常名;
```

3. 处理异常

触发异常处理后，可以定义异常处理部分，语法如下：

```
Exception
When 异常名 1 then
    异常处理语句段 1;
When 异常名 2 then
    异常处理语句段 2;
```

4. 实例

下面的 PL/SQL 程序包含了完整的异常处理定义、触发、处理的过程。定义名为 salaryerror 的异常，在 scott.emp 数据表中查找 empno=7566 的记录，将其值放入变量 tempsal 中，判断 tempsal 值若不在 900 和 2600 之间，说明该员工的薪水有问题，将激活异常处理，提示信息。

在【SQLPlus Worksheet】中执行下列 PL/SQL 代码，执行结果如图 9.57 所示。

```

set serveroutput on
declare
    salaryerror exception;
    tempsal scott.emp.sal%type;
begin
    select sal into tempsal
    from scott.emp
    where empno=7566;
    if tempsal<900 or tempsal>2600 then
        raise salaryerror;
    end if;
exception
    when salaryerror then
        dbms_output.put_line('薪水超出范围');
end;

```

【配套程序位置】：第9章\exceptiondefine.sql。



图 9.57 自定义异常

9.9 综合实例

本节将在前面学习的 PL/SQL 基础上分析一个较复杂的实例，以教会读者编写完整的 PL/SQL 程序的方法。

9.9.1 实例设计

1. 功能设计

某高校开发的研究生招生系统，要求设计 PL/SQL 程序对考生的成绩数据进行处理，处理的逻辑是根据每门专业课的最低分数线和总分的最低分数线自动将考生归类为录取考生、调剂考生和落选考生。

为此设计两个数据表，graduate 数据表存放考生成绩，result 数据表存放处理结果，PL/SQL 程序完成的功能就是将 graduate 数据表中的数据逐行扫描，根据分数线进行判断，计算出各科总分，在 result 数据表中将标志字段自动添加上“录取”或“落选”。

2. 数据表设计

Graduate 数据表结构如表 9.3 所示。

表 9.3 graduate 数据表结构

字段名称	类型	宽度	小数位数	说明
BH	NUMBER	9		考生编号
XM	VARCHAR2	9		考生姓名
LB	VARCHAR2	9		报考类别
YINGYU	NUMBER	4	1	英语成绩
ZHENGZHI	NUMBER	4	1	政治成绩
ZHUANYE1	NUMBER	4	1	专业课 1
ZHUANYE2	NUMBER	4	1	专业课 2
ZHUANYE3	NUMBER	4	1	专业课 3

result 数据表结构如表 9.4 所示。

表 9.4 result 数据表结构

字段名称	类型	宽度	小数位数	说明
BH	NUMBER	9		考生编号
XM	VARCHAR2	9		考生姓名
LB	VARCHAR2	9		报考类别
YINGYU	NUMBER	4	1	英语成绩
ZHENGZHI	NUMBER	4	1	政治成绩
ZHUANYE1	NUMBER	4	1	专业课 1
ZHUANYE2	NUMBER	4	1	专业课 2
ZHUANYE3	NUMBER	4	1	专业课 3

续表

字段名称	类型	宽度	小数位数	说明
TOTALSCORE	NUMBER	5	1	总分
FLAG	VARCHAR2	4		标志

9.9.2 创建数据表

为了简化起见，实例的两个数据表都建立在默认的 scott 用户下，这样读者可以完全模拟实例一致的环境进行学习。

1. 创建 graduate 数据表

在图 9.58 所示的创建表的【一般信息】选项卡中按照下列步骤进行配置。

在【名称】文本框中输入“GRADUATE”。

在【方案】下拉列表框中选择“SCOTT”。

在【表空间】下拉列表框中选择“USERS”。

在【表结构定义区】按照图所示进行设置。


完成设置后单击  按钮。



图 9.58 创建表的【一般信息】选项卡

读者也可以在【SQLPlus Worksheet】中执行下列 SQL 代码，直接创建数据表 SCOTT.GRADUATE。

```
CREATE TABLE "SCOTT"."GRADUATE" (  
    "BH"          NUMBER(10)          NOT NULL,  
    "XM"          VARCHAR2(10)        NOT NULL,  
    "LB"          VARCHAR2(10)        NOT NULL,
```

```

"YINGYU"    NUMBER(4, 1)    NOT NULL,
"ZHENGZHI"  NUMBER(4, 1)    NOT NULL,
"ZHUANYE1"  NUMBER(4, 1)    NOT NULL,
"ZHUANYE2"  NUMBER(4, 1)    NOT NULL,
"ZHUANYE3"  NUMBER(4, 1)    NOT NULL)

```

TABLESPACE "USERS"

【配套程序位置】：第 9 章\creategraduate.sql。

2. 创建 result 数据表

在图 9.59 所示的创建表的【一般信息】选项卡中按照下列步骤进行配置。

在【名称】文本框中输入“RESULT”。

在【方案】下拉列表框中选择“SCOTT”。

在【表空间】下拉列表框中选择“USERS”。

在【表结构定义区】按照图所示进行设置。


完成设置后单击  按钮。



图 9.59 创建表的【一般信息】选项卡

读者也可以在【SQLPlus Worksheet】中执行下列 SQL 代码，直接创建数据表 SCOTT.RESULT。

```

CREATE TABLE "SCOTT"."RESULT" (
    "BH"          NUMBER(10)    NOT NULL,
    "XM"          VARCHAR2(10)  NOT NULL,
    "LB"          VARCHAR2(10)  NOT NULL,
    "YINGYU"      NUMBER(4, 1)  NOT NULL,

```



```

"ZHENGZHI" NUMBER(4, 1) NOT NULL,
"ZHUANYE1" NUMBER(4, 1) NOT NULL,
"ZHUANYE2" NUMBER(4, 1) NOT NULL,
"ZHUANYE3" NUMBER(4, 1) NOT NULL,
"TOTALSCORE" NUMBER(5, 1) NOT NULL,
"FLAG" VARCHAR2(4) NOT NULL)
TABLESPACE "USERS"

```

【配套程序位置】：第 9 章\createreult.sql。

9.9.3 录入数据

利用【表数据编辑器】向 SCOTT.GRADUATE 数据表录入下列实验数据。如图 9.60 所示。

图 9.60 向 graduate 数据表录入测试数据

读者也可以在【SQLPlus Worksheet】中执行下列 SQL 代码，直接向数据表 SCOTT.GRADUATE 中录入测试数据。

```

INSERT INTO "SCOTT"."GRADUATE"
("BH", "XM", "LB", "YINGYU", "ZHENGZHI", "ZHUANYE1", "ZHUANYE2", "ZHUANYE3")
VALUES (2003080520, '张三丰', '硕士', 55, 56, 67, 78, 89);
INSERT INTO "SCOTT"."GRADUATE"
("BH", "XM", "LB", "YINGYU", "ZHENGZHI", "ZHUANYE1", "ZHUANYE2", "ZHUANYE3")
VALUES (2003060555, '张翠山', '硕士', 66, 78, 78, 89, 92);
INSERT INTO "SCOTT"."GRADUATE"
("BH", "XM", "LB", "YINGYU", "ZHENGZHI", "ZHUANYE1", "ZHUANYE2", "ZHUANYE3")
VALUES (2003056066, '张无忌', '硕士', 76, 67, 89, 90, 66);
INSERT INTO "SCOTT"."GRADUATE"
("BH", "XM", "LB", "YINGYU", "ZHENGZHI", "ZHUANYE1", "ZHUANYE2", "ZHUANYE3")
VALUES (2003010989, '赵敏', '硕士', 45, 59, 74, 66, 56);
INSERT INTO "SCOTT"."GRADUATE"

```

```

("BH", "XM", "LB", "YINGYU", "ZHENGZHI", "ZHUANYE1", "ZHUANYE2", "ZHUANYE3" )
VALUES (2003050677, '周芷若', '硕士', 77, 67, 72, 87, 66 );
INSERT INTO "SCOTT"."GRADUATE"
("BH", "XM", "LB", "YINGYU", "ZHENGZHI", "ZHUANYE1", "ZHUANYE2", "ZHUANYE3" )
VALUES (2003869401, '小昭', '硕士', 56, 67, 56, 64, 34 );
INSERT INTO "SCOTT"."GRADUATE"
("BH", "XM", "LB", "YINGYU", "ZHENGZHI", "ZHUANYE1", "ZHUANYE2", "ZHUANYE3" ) VALUES
(2003340987, '阿离', '硕士', 68, 93, 64, 80, 56 );
INSERT INTO "SCOTT"."GRADUATE"
("BH", "XM", "LB", "YINGYU", "ZHENGZHI", "ZHUANYE1", "ZHUANYE2", "ZHUANYE3" )
VALUES (2003056709, '宋远桥', '硕士', 90, 68, 81, 61, 67 );
INSERT INTO "SCOTT"."GRADUATE"
("BH", "XM", "LB", "YINGYU", "ZHENGZHI", "ZHUANYE1", "ZHUANYE2", "ZHUANYE3" ) VALUES
(2003100894, '殷素素', '硕士', 69, 73, 62, 70, 75 );

```

【配套程序位置】：第 9 章\insertgraduate.sql。

9.9.4 程序设计

1. 创建处理过程 scott.graduateprocess

在【SQLPlus Worksheet】中执行下列 PL/SQL 代码，创建处理过程 scott.graduateprocess。执行结果如图 9.61 所示。



图 9.61 创建处理过程 scott.graduateprocess

```
/*定义过程参数*/
```

```
create or replace procedure scott.graduateprocess(
    tempzhengzhi in scott.graduate.zhengzhi%type,
```

```

    tempyingyu in scott.graduate.yingyu%type,
    tempzhuanyel in scott.graduate.zhuanyel%type,
    tempzhuanye2 in scott.graduate.zhuanye2%type,
    tempzhuanye3 in scott.graduate.zhuanye3%type,
    temptotalscore in scott.result.totalscore%type) as
/*定义graduaterecord为记录型变量，临时存放通过游标从graduate数据表中提取的记录*/
    graduaterecord scott.graduate%rowtype;
/*定义graduatetotalscore为数值型变量，统计总分*/
    graduatetotalscore scott.result.totalscore%type;
/*定义graduateflag为字符型变量，根据结果放入“落选”或“录取”，然后写入数据表result*/
    graduateflag varchar2(4);
/*定义游标graduatecursor，存放的是所有的graduate数据表中的记录*/
    cursor graduatecursor is
        select * from scott.graduate;
/*定义异常处理errormessage*/
    errormessage exception;
/*开始执行*/
begin
    /*打开游标*/
    open graduatecursor;
/*如果游标没有数据，激活异常处理*/
    if graduatecursor%notfound then
        raise errormessage;
    end if;
/*游标有数据，指针指向第一条记录，每执行fetch命令，就自动下移，循环执行到记录提取完毕为止
*/
    loop
        fetch graduatecursor into graduaterecord;
/*计算总分*/
        graduatetotalscore:=graduaterecord.yingyu+graduaterecord.zhengzhi+graduaterecord.zhuanye
1+graduaterecord.zhuanye2+graduaterecord.zhuanye3;
/*判断单科和总分是否满足录取要求，若满足，graduateflag变量值为“录取”，否则为“落选”*/
    if (graduaterecord.yingyu>=tempyingyu and
        graduaterecord.zhengzhi>=tempzhengzhi and
        graduaterecord.zhuanyel>=tempzhuanyel and
        graduaterecord.zhuanye2>=tempzhuanye2 and
        graduaterecord.zhuanye3>=tempzhuanye3 and
        graduatetotalscore>=temptotalscore) then
        graduateflag:='录取';
    else

```

```

        graduateflag:='落选';
    end if;
    /*当游标数据提取完毕后，退出循环*/
    exit when graduatecursor%notfound;
    /*向结果数据表result中插入处理后的数据*/
    insert into
scott.result(bh, xm, lb, zhengzhi, yingyu, zhuaney1, zhuaney2, zhuaney3, totalscore, flag)
values(graduaterecord. bh, graduaterecord. xm, graduaterecord. lb, graduaterecord. zhengzhi, graduate
record. yingyu, graduaterecord. zhuaney1, graduaterecord. zhuaney2, graduaterecord. zhuaney3, graduat
etotalscore, graduateflag);
    end loop;
    /*关闭游标*/
    close graduatecursor;
    /*提交结果*/
    commit;
    /*异常处理，提示错误信息*/
    exception
    when errormessage then
        dbms_output.put_line('无法打开数据表');
    /*程序执行结束*/
end;

```

【配套程序位置】：第9章\creategraduateprocess.sql。

2. 主程序 mainprocess 设计

主程序调用名为 graduateprocess 的过程来完成处理，代码设计如下：

```

set serveroutput on
/*定义6个入口变量，分别对应graduate数据表中的专业课和总分分数线*/
declare
    score1 number(4,1);
    score2 number(4,1);
    score3 number(4,1);
    score4 number(4,1);
    score5 number(4,1);
    scoretotal number(5,1);
    /*将分数线赋值，在这里修改各值就代表不同的分数线*/
begin
    score1:=50;

```

```

score2:=56;
score3:=60;
score4:=62;
score5:=64;
scoretot:=325;
/*调用处理过程*/
scott.graduateprocess(score1, score2, score3, score4, score5, scoretot);
end;

```

【配套程序位置】：第9章\mainprocess.sql。

在上述过程设计中，综合使用了异常处理、游标、变量等 PL/SQL 程序设计要素，通过主程序向带参数执行的过程传递参数。

9.9.5 执行结果

1. 第一组执行结果

(1) 将 mainprocess.sql 文件稍做改动，在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，执行结果如图 9.62 所示。

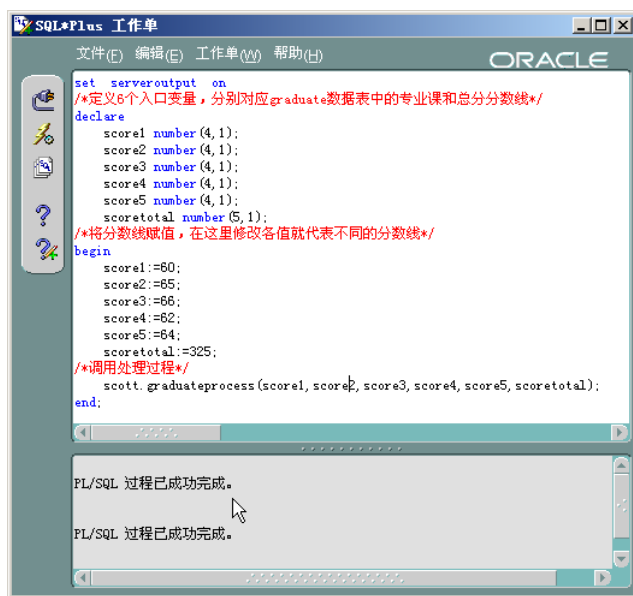


图 9.62 进行第一组测试

```

set serveroutput on
declare
  score1 number(4,1);

```

```

score2 number(4,1);
score3 number(4,1);
score4 number(4,1);
score5 number(4,1);
scoretototal number(5,1);

begin
score1:=60;
score2:=65;
score3:=66;
score4:=62;
score5:=64;
scoretototal:=325;

scott.graduateprocess(score1,score2,score3,score4,score5,scoretototal);

end;

```

【配套程序位置】：第 9 章\mainprocess1.sql。

(2) 利用【表数据编辑器】查看 scott.result 数据表的结果，如图 9.63 所示。

BH	XM	LB	YIN...	ZHE...	ZHU...	ZHU...	ZH...	TO...	FLAG
2003080520	张三丰	硕士	55	56	67	78	89	345	落选
2003060555	张翠山	硕士	66	78	78	89	92	403	录取
2003056066	张无忌	硕士	76	67	89	90	66	388	录取
2003010989	赵敏	硕士	45	59	74	66	56	300	落选
2003050677	周芷若	硕士	77	67	72	87	66	369	录取
2003869401	小昭	硕士	56	67	56	64	34	277	落选
2003340987	阿离	硕士	68	93	64	80	56	361	落选
2003056709	宋远桥	硕士	90	68	81	61	67	367	落选
2003100894	殷素素	硕士	69	73	62	70	75	349	落选

图 9.63 第一组参数执行结果

2. 第二组执行结果

(1) 将 mainprocess.sql 文件稍做改动，在【SQLPlus Worksheet】中执行下列 PL/SQL 程序，执行结果如图 9.64 所示。

```

set serveroutput on
declare
score1 number(4,1);
score2 number(4,1);
score3 number(4,1);
score4 number(4,1);
score5 number(4,1);

```

```

scoretotal number(5,1);
begin
    score1:=65;
    score2:=64;
    score3:=62;
    score4:=70;
    score5:=65;
    scoretotal:=335;
    scott.graduateprocess(score1,score2,score3,score4,score5,scoretotal);
end;

```

【配套程序位置】：第9章\mainprocess2.sql。

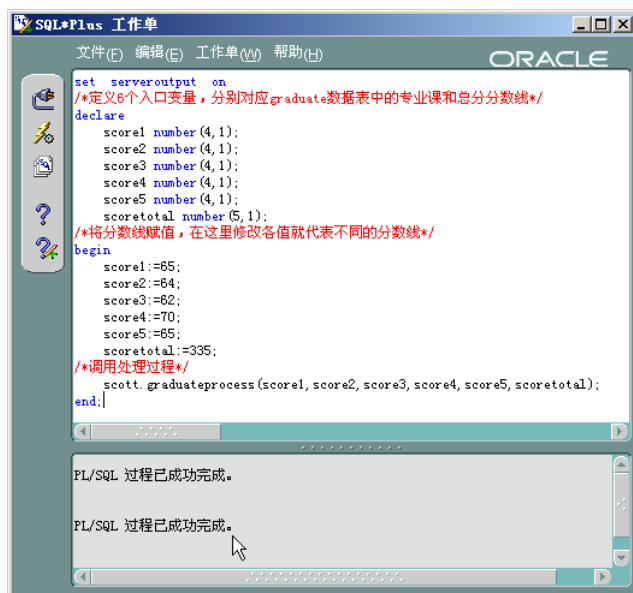


图 9.64 进行第二组测试

(2) 利用【表数据编辑器】查看 scott.result 数据表的结果，如图 9.65 所示。

BH	XM	LB	YIN...	ZH...	ZH...	ZH...	ZH...	TOT...	FLAG
2003080520	张三丰	硕士	55	56	67	78	89	345	落选
2003060555	张翠山	硕士	66	78	78	89	92	403	录取
2003056066	张无忌	硕士	76	67	89	90	66	388	录取
2003010989	赵敏	硕士	45	59	74	66	56	300	落选
2003050677	周芷若	硕士	77	67	72	87	66	369	录取
2003889401	小昭	硕士	56	67	56	64	34	277	落选
2003340987	阿离	硕士	68	93	64	80	56	361	落选
2003056709	宋远桥	硕士	90	68	81	61	67	367	落选
2003100894	殷素素	硕士	69	73	62	70	75	349	录取

图 9.65 第一组参数执行结果

综合运用 PL/SQL 的设计要素，就可以设计出很多复杂的处理程序，这也是 DBA 的一项重要任务。

9.10 习题

- (1) PL/SQL 程序的主要特点是什么？
- (2) 如何定义常量？
- (3) PL/SQL 中支持哪些基本的数据类型变量？试编写测试 PL/SQL 基本数据类型变量的实验代码。
- (4) 如何定义常用的复合类型数据变量？
- (5) PL/SQL 程序中有哪些常见的表达式？
- (6) 什么是过程？过程的特点是什么？
- (7) 什么是序列？如何使用序列？
- (8) 如何自定义异常处理？在 PL/SQL 程序中为什么要定义异常处理？
- (9) 试编写一个完整的 PL/SQL 程序，自行设计数据表结构。要求在 PL/SQL 程序中综合运用序列、过程、异常处理、表达式、函数、常量、变量、流程控制和事务处理的语句。写出实验结果和代码。