第 3 章

控制结构

单元概述

本章将介绍 PL/SQL 中的控制语句，包括条件分支语句、循环语句。条件分支语句根据不同的条件进行不用的逻辑处理，循环语句根据不同的循环条件执行相同的循环体。条件分支语句和循环语句是 PL/SQL 过程化的重要特性。通过本章学习，读者可以掌握 PL/SQL中条件分支语句与循环语句的应用。

教学重点与难点

重点：

(1)条件分支语句的使用。

(2)循环语句的使用。

难点：

(1)条件分支语句中 NULL 值的处理。

(2)FOR 循环语句的应用。

3.1 条件分支语句

在 PL/SQL 中，条件分支语句包括 IF 语句和 CASE 语句两种。

3.1.1 IF 语句

利用 IF 语句实现分支选择的语法为：

```
IF condition1 THEN
   statements1;
[ELSIF condition2 THEN
   statements2;]
……
[ELSE
   else_statements];
END IF;
```

IF-THEN-ELSE 语句执行流程如图 3-1 所示，IF-THEN-ELSIF 语句执行流程如图 3-2 所示。

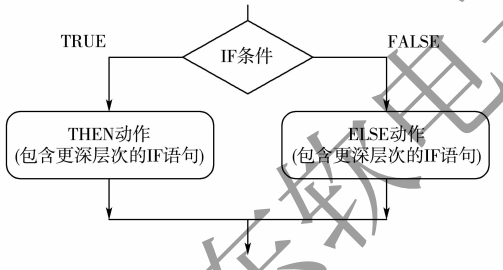

图 3-1 IF-THEN-ELSE 语句

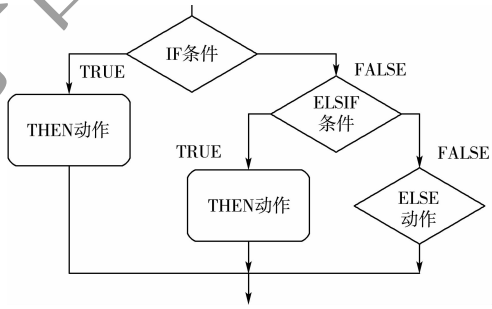

图 3-2 IF-THEN-ELSIF 语句

例如，输入一个员工号，修改该员工的工资，如果该员工为 10 号部门，工资增加 100；若为 20 号部门，工资增加 150；若为 30 号部门，工资增加 200；否则增加 300。

```
DECLARE
   v_deptno emp.deptno%TYPE;
   v_increment NUMBER(4);
   v_empno   emp.empno%TYPE :=7844;
BEGIN
   SELECT deptno INTO v_deptno FROM emp WHERE empno=v_empno;
   IF v_deptno=10 THEN v_increment:=100;
   ELSIF v_deptno=20 THEN v_increment:=150;
   ELSIF v_deptno=30 THEN v_increment:=200;
   ELSE  v_increment:=300;
   END IF;
   UPDATE emp SET sal=sal+v_increment WHERE empno=v_empno;
END;
```

IF 语句可以相互嵌套，即在一个 IF 语句内部包含另一个 IF 语句。例如，上例还可以按下列语句执行：

```
DECLARE
   v_deptno emp.deptno%TYPE;
   v_increment NUMBER(4);
   v_empno   emp.empno%TYPE :=7844;
BEGIN
   SELECT deptno INTO v_deptno FROM emp WHERE empno=v_empno;
   IF v_deptno=10 THEN
      v_increment:=100;
   ELSE
      IF v_deptno=20 THEN
         v_increment:=150;
      ELSE
          IF v_deptno=30 THEN
             v_increment:=200;
          ELSE
             v_increment:=300;
          END IF;
      END IF;
   END IF;
   UPDATE emp SET sal=sal+v_increment WHERE empno=v_empno;
END;
```

3.1.2　分支条件中的 NULL 处理

由于 PL/SQL 中逻辑运算结果有 TRUE、FALSE 和 NULL 三种，因此在进行分支条件判断时，需要考虑分支条件为 NULL 的情况。

(1)数值与 NULL 比较

如果分支条件表达式是一个数值与 NULL 值的比较，那么其逻辑结果为 NULL。

例如，数值与 NULL 比较示例。

```
DECLARE
   v_x number(2):= 5;
   v_y number(2):= NULL;
BEGIN
   IF v_x != v_y THEN
      DBMS_OUTPUT.PUT_LINE(100);
   ELSE
      DBMS_OUTPUT.PUT_LINE (200);
   END IF;
END;
/
200
PL/SQL 过程已成功完成。
```

(2)NULL 与 NULL 的比较

如果分支条件表达式是两个 NULL 值的比较,那么其逻辑结果为 NULL。

例如,两个 NULL 比较示例。

```
DECLARE
   v_x number(2):= NULL;
   v_y number(2):= NULL;
BEGIN
   IF v_x = v_y THEN
     DBMS_OUTPUT.PUT_LINE(100);
   ELSE
     DBMS_OUTPUT.PUT_LINE(200);
   END IF;
END;
/
200
PL/SQL 过程已成功完成。
```

(3)NULL 转换后的处理

如果分支条件表达式中包含 NULL 值,可以将 NULL 值转换后再进行处理。

例如,NULL 值转换后的比较示例。

```
DECLARE
  v_x number(2):= 5;
  v_y number(2):= NULL;
BEGIN
  IF nvl(v_x,0) ! = nvl(v_y,0) THEN
    DBMS_OUTPUT.PUT_LINE(100);
  ELSE
    DBMS_OUTPUT.PUT_LINE(200);
  END IF;
END;
/
100
PL/SQL 过程已成功完成。
```

(4)使用 IS NULL 或 IS NOT NULL 进行 NULL 的判断

可以使用 IS NULL 或 IS NOT NULL 进行 NULL 值的判断。例如:

```
DECLARE
  v_x number(2):= NULL;
BEGIN
  IF v_x IS NULL THEN
     DBMS_OUTPUT.PUT_LINE (100);
  ELSE
    DBMS_OUTPUT.PUT_LINE (200);
```

```
  END IF;
END;
/
100
PL/SQL 过程已成功完成。
```

由上述的几个例子可以看出，如果在分支条件表达式中包含 NULL，那么任何值与 NULL 进行关系运算的结果都为 NULL。如果程序的处理逻辑没有考虑到 NULL 的情况，程序运行容易出现歧异。为了避免分支条件为 NULL 时出现歧义，应该在程序中进行条件是否为 NULL 的检查，对条件为 NULL、TRUE 和 FALSE 三种情况都进行处理。

例如，分支条件中包含 NULL 的处理示例。

```
DECLARE
   v_x number(2):= 5;
   v_y number(2):= NULL;
BEGIN
   IF v_x IS NULL OR v_y IS NULL THEN
     DBMS_OUTPUT.PUT_LINE('UNKNOW');
   ELSIF v_x ! = v_y THEN
     DBMS_OUTPUT.PUT_LINE(100);
   ELSE
     DBMS_OUTPUT.PUT_LINE (200);
   END IF;
END;
/
UNKNOW
PL/SQL 过程已成功完成。
```

在分支条件表达式中，经常进行布尔型变量(TRUE、FALSE、NULL)的逻辑运算。

逻辑运算符包括 AND(逻辑与)、OR(逻辑或)和 NOT(逻辑非)，其中 AND 与 OR 是二元运算符，NOT 是一元运算符。它们之间的逻辑运算真值如表 3-1 所示。

表 3-1　　逻辑运算真值表

x	y	x AND y	x OR y	NOT x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	TRUE
NULL	TRUE	NULL	TRUE	NULL
NULL	FALSE	FALSE	NULL	NULL
NULL	NULL	NULL	NULL	NULL

3.1.3 CASE 语句

CASE 语句可以根据条件从多个执行分支中选择相应的执行动作，并能返回一个值。

CASE 语句有由两种形式，一种只进行等值比较，另一种进行多种条件的比较。

(1)只进行等值比较的 CASE 语句

只进行等值比较的 CASE 语句的语法为：

```
CASE selector
   WHEN expression1 THEN result1
   WHEN expression2 THEN result2
   ...
   WHEN expressionN THEN resultN
   [ELSE resultN+1]
END;
```

CASE 语句判断 expression 表达式的值是否与 selector 相等。如果相等，则执行其后的语句；如果任何 expression 表达式的值与 selector 值都不等，则执行 ELSE 后的 resultN+1 语句。

例如，查询并输出工资最高的员工所在的部门。语句为：

```
DECLARE
   v_var VARCHAR2(10);
   v_deptno emp.deptno%TYPE;
BEGIN
   SELECT deptno INTO v_deptno FROM emp WHERE sal = (SELECT max(sal) FROM emp);
   v_var :=
   CASE v_deptno
      WHEN 10 THEN '部门一'
      WHEN 20 THEN '部门二'
      ELSE '部门三'
   END;
   DBMS_OUTPUT.PUT_LINE(v_var );
END;
/
部门一
PL/SQL 过程已成功完成。
```

(2)可以进行多种条件比较的 CASE 语句

利用 CASE 语句可以进行多种条件的比较，其语法为：

```
CASE
   WHEN condition1 THEN statements1;
   WHEN condition2 THEN statements2;
   …
   WHEN conditionn THEN statementsn;
   [ELSE    else_statements;]
END CASE;
```

CASE 语句对每一个 WHEN 条件进行判断，当条件为真时，执行其后的语句。如果所有条件都不为真，则执行 ELSE 后的语句。

例如，求雇员的平均工资，当工资小于 1000 时，提示“工资太低”；当工资大于等于 1000 小于 2000 时，提示“起步的工资”；当工资大于等于 2000 时，提示“已经步入软件行业中”。使用 CASE 表达式来实现。

```
DECLARE
   v_var VARCHAR2(20);
   v_sal emp.sal%TYPE;
BEGIN
   SELECT avg(sal) INTO v_sal FROM emp;
   v_var :=
   CASE
     WHEN v_sal <1000 THEN '工资太低'
     WHEN v_sal>=1000 AND v_sal <2000 THEN '起步的工资'
     ELSE '已经步入软件行业中'
   END;
   DBMS_OUTPUT.PUT_LINE(v_var);
END;
/
'已经步入软件行业中'
PL/SQL 过程已成功完成。
```

注意，在 CASE 语句中，当第一个 WHEN 条件为真时，执行其后的操作，操作完后结束 CASE 语句。其他的 WHEN 条件不再判断，其后操作也不执行。

3.2　循环语句

在 PL/SQL 中，循环结构有三种形式，分别为无条件的循环（简单循环）、有计数的循环（FOR 循环）和有条件的循环（WHILE 循环）三种。

3.2.1　无条件循环

无条件循环的特点是循环体至少执行一次，LOOP 语句的语法为：

```
LOOP
  statement1;
  …
  EXIT [WHEN condition];
END LOOP;
```

在使用 LOOP 语句时应该使用 EXIT 语句，强制循环结束，否则将发生死循环。

例如，利用无条件循环向 test 表中插入 10 条记录。语句为：

```
DECLARE
  v_count number(2):= 1;
  v_empno emp.empno%type;
  v_ename emp.ename%type:='SMITH';
  v_job emp.job%type:='MANAGER';
BEGIN
  SELECT max(empno) INTO v_empno
  FROM emp;
  LOOP
    INSERT INTO test(empno,ename,job)
    VALUES ((v_empno+v_count),v_ename,v_job);
    v_count := v_count + 1;
    EXIT WHEN v_count >10;
  END LOOP;
END;
```

3.2.2 FOR 循环

在无条件循环中，需要定义循环变量，不断修改循环变量的值，以达到控制循环次数的目的，而在 FOR 循环中，不需要显式定义循环变量，系统自动定义一个循环计数器，每次循环时该循环计数器的值自动增 1 或减 1，以控制循环的次数。

FOR 循环的语法为：

```
FOR counter in [REVERSE]lower_bound..upper_bound LOOP
  statement1;
  statement2;
  ...
END LOOP;
```

其中，counter 为循环计数器，lower_bound 为循环计数器的下界（最小值），upper_bound 为循环计数器的上界（最大值）。

使用 FOR 循环，需要注意下列事项：

- 循环计数器不需要显式定义，系统隐含地将它声明为 BINARY_INTEGER 变量。
- 系统默认时，循环计数器从下界往上界递增计数，如果使用 REVERSE 关键字，则表示循环计数器从上界向下界递减计数。
- 循环计数器只能在循环体中使用，不能在循环体外使用。

例如，利用 FOR 循环向 test 表中插入 10 条记录。语句为：

```
DECLARE
   v_empno emp.empno%type;
   v_ename emp.ename%type := 'ljs';
   v_job emp.job%type := 'manager';
BEGIN
   SELECT max(empno) INTO v_empno FROM emp;
```

```
    FOR v_count IN 1..10 LOOP
      INSERT INTO test(empno,ename,job)VALUES ((v_empno+v_count),v_ename,v_job);
    END LOOP;
END;
```

3.2.3　WHILE 循环

利用 WHILE 语句进行循环时，先判断循环条件，只有满足循环条件才能进入循环体进行循环操作，其语法为：

```
WHILE condition LOOP
  statement1;
  statement2;
  …
END LOOP;
```

例如，利用 WHILE 循环向 test 表中插入 10 条记录。语句为：

```
DECLARE
  v_count number(2):= 1;
  v_empno emp.empno%type;
  v_ename emp.ename%type:='ljs';
  v_job emp.job%type:='manager';
BEGIN
  SELECT max(empno) INTO v_empnoFROM emp;
  WHILE v_count <= 10 LOOP
    INSERT INTO test(empno,ename,job) VALUES ((v_empno+v_count),v_ename,v_job);
    v_count := v_count + 1;
  END LOOP;
END;
```

3.3　课后作业

1.思考题

(1)PL/SQL 中的 IF 语句和 Java 中的 if 语句在作用和语法上有什么不同？

(2)ELSIF 和 ELSE IF 子句有什么不同？

(3)CASE 语句和 Java 中哪个语句的作用相同？和这个语句的不同点又在哪？

(4)CASE 语句有哪两种应用？

(5)无条件循环的执行流程是什么？和 Java 中哪种循环类似？

(6)使用无条件循环应该注意的问题包括哪些？

(7)FOR 循环的执行流程。

(8)什么时候使用 FOR 循环？

(9)使用 FOR 循环应该注意的问题包括哪些？

(10)WHILE 循环执行的流程。

(11)使用 WHILE 循环应该注意的问题?

2.按下列要求编写 PL/SQL 程序

(1)写 SQL 语句,向部门表中添加一个字段 maxnumber 整型,表示部门编制人数。

(2)把 10 号部门的编制更新为 5 人。

(3)写一个块,用来向 10 号部门入职一名新员工,员工编号为当前最大员工编号加 1,员工姓名为 TOM,岗位为 CLERK,其他字段都为 null;当 10 号部门最大的人数不超过编制人数时,入职成功;当部门的人数超过编制人数时,提示入职失败。

(4)使用简单循环,批量入职 5 名员工,员工编号分别为当前最大编号加 1,部门为 20 号部门,姓名为 zs1,zs2…,入职日期为当前日期,其他字段为 NULL,暂时不判断人数是否超编。

(5)使用 FOR 循环,遍历员工信息,依次输出每个员工的姓名及部门名称。

(6)使用 WHILE 循环,遍历员工信息,依次输出每个员工的姓名及部门名称。

(7)使用外部替代变量提供雇员的 ID,传递该值到 PL/SQL 块,查询 emp 表的薪水,如果薪水小于 2000 的,显示“挣的不多,需努力”;如果薪水在 2000 到 5000 的,显示“收入还可以,还需努力”;薪水大于 5000 的显示“挣的挺多了,歇歇吧”。

(8)屏幕上输出 1 到 10(不包括 6 和 8)。

(9)使用 FOR 循环和 WHILE 循环,分别实现练习(4)的批量员工入职功能。