第3章　数据库编程

标准 SQL 是非过程化的查询语言，具有操作统一、面向集合、功能丰富、使用简单等多项优点。但和程序设计语言相比，高度非过程化的优点同时也造成了它的一个弱点：缺少流程控制能力，难以实现应用业务中的逻辑控制。SQL 编程技术可以有效克服 SQL 实现复杂应用方面的不足，提高应用系统和 RDBMS 间的互操作性。

这里主要介绍 Oracle 19c 中与数据库编程相关的内容。

3.1　PL/SQL 编程的基础

PL/SQL 是 Oracle 的专用语言，它是对标准 SQL 的扩展。SQL 语句可以嵌套在 PL/SQL 代码中，将 SQL 的数据处理能力和 PL/SQL 的过程处理能力结合在一起。在 Oracle 数据库以及开发工具中都内置了 PL/SQL 处理引擎。PL/SQL 被集成在 Oracle 数据库服务器产品中，因此，其代码可以得到非常高效的处理。

3.1.1　PL/SQL 程序的结构

PL/SQL 程序的基本结构是块。所有的 PL/SQL 程序都是由块组成的。这些块之间可以互相嵌套，每个块完成一个逻辑操作。

PL/SQL 程序通常包括 3 部分：

(1) DECLARE 部分。DECLARE 部分包含定义变量、常量和游标等类型的代码。

(2) BEGIN…END 部分。BEGIN…END 部分是程序的主体，其中还可以再嵌套 BEGIN…END 部分。该部分包含了该程序块的所有处理操作。

(3) EXCEPTION 部分。EXCEPTION 部分是异常处理部分，允许在执行 BEGIN…END 部分发生异常时控制程序的执行。

一个程序块总是以 END 语句结束的，其中 BEGIN…END 部分是 PL/SQL 必需的部分，但一般的程序块都包括这 3 部分，其基本结构如图 3-1 所示。

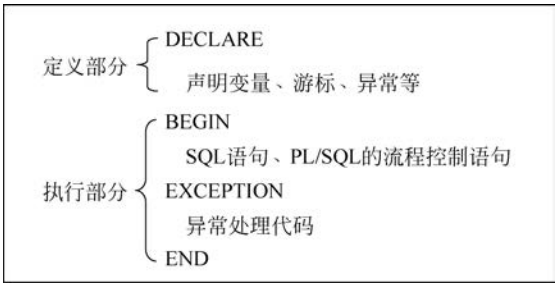

图 3-1 PL/SQL 程序块的基本结构

【例 3-1】 PL/SQL 程序块示例。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  sum_num number(2);
  3  BEGIN
  4  SELECT COUNT( * ) INTO sum_num FROM dept;
  5  dbms_output.put_line('记录个数:'||sum_num);
  6  END;
  7  /
记录个数:4
PL/SQL 过程已成功完成。
```

注意：SET SERVEROUTPUT ON 为打开服务器的输出显示，即打开 Oracle 自带的输出方法 dbms_output。

【例 3-2】 PL/SQL 程序块的嵌套使用。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE                    -- 外层程序块头
  2   out_text1 varchar2(20):='外层程序块';
  3  BEGIN
  4   DECLARE                   -- 内层程序块头
  5    out_text2 varchar2(20);
  6   BEGIN
  7    out_text2:='内层程序块';
  8    dbms_output.put_line(out_text2);
  9   END;                      -- 内层程序块尾
 10   dbms_output.put_line(out_text1);
 11  END;                       -- 外层程序块尾
 12  /
内层程序块
外层程序块
PL/SQL 过程已成功完成。
```

注意：变量可以在程序块的 DECLARE 部分和 BEGIN…END 部分为其赋值。赋值时，常用的方法是使用 PL/SQL 赋值操作符“:=”。

3.1.2 使用%TYPE 和%ROWTYPE 类型的变量

在定义变量时，除了可以使用 Oracle 规定的数据类型外，还可以使用%TYPE 和%ROWTYPE 来定义变量。

1. %TYPE 变量

在例 3-1 中,为了存储从数据库中检索到的数据,首先根据检索的数据列的数据类型定义变量,然后使用 SELECT 语句中的 INTO 子句将检索到的数据保存到变量中。这里有一个前提条件,即用户必须事先知道检索的数据类型。如果用户事先并不知道检索的数据列的数据类型,这时可以考虑使用%TYPE 定义变量。

【例 3-3】 使用%TYPE 变量类型。

```
SQL > SET SERVEROUTPUT ON
SQL > DECLARE
  2    no   dept.deptno % type;
  3    name dept.dname % type;
  4    place dept.loc % type;
  5   BEGIN
  6    SELECT deptno,dname,loc
  7    INTO no,name,place
  8    FROM dept WHERE deptno = 10;
  9    dbms_output.put_line(no||''||name||''||place);
 10   END;
 11   /
10 ACCOUNTING NEW YORK
PL/SQL 过程已成功完成。
```

使用%TYPE 定义变量的好处如下:

(1) 用户不必查看数据类型就可以确保定义的变量能够存储检索的数据。

(2) 使用%TYPE 类型的变量后,如果用户后期修改数据库结构(如改变某列的数据类型),则不必考虑对所定义的变量进行更改。

2. %ROWTYPE 变量

%ROWTYPE 类型的变量一次可以存储从数据库检索的一行数据。该变量的结构与检索表的结构完全相同。

【例 3-4】 使用%ROWTYPE 变量类型。

```
SQL > SET SERVEROUTPUT ON
SQL > DECLARE
  2    row_dept dept % ROWTYPE;
  3   BEGIN
  4    SELECT *
  5    INTO row_dept
  6    FROM dept WHERE deptno = 10;
  7    dbms_output.put_line(row_dept.deptno);
  8    dbms_output.put_line(row_dept.dname||''||row_dept.loc);
  9   END;
 10   /
10 ACCOUNTING NEW YORK
PL/SQL 过程已成功完成。
```

3.1.3 条件判断语句

PL/SQL 与其他编程语言一样,也都具有条件判断语句。条件判断语句主要的作用是根据条件的变化选择执行不同的代码。PL/SQL 中常用的条件判断语句有 IF 语句和 CASE 语句。

1. IF 语句

在 PL/SQL 中,为了控制程序的执行方向,引进了 IF 语句。IF 语句主要有如下两种形式。

1) 形式一

```
IF 条件 THEN
  PL/SQL 语句 1 或 SQL 语句 1;
[ ELSE
  PL/SQL 语句 2 或 SQL 语句 2;
]
END IF;
```

ELSE 短语用方括号括起来,同其他语言一样,表示它为可选项。

【例 3-5】 IF 语句示例 1:判断变量 a 和 b 的大小。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
   2   a number;
   3   b number;
   4  BEGIN
   5   a:= 1;
   6   b:= 2;
   7   IF a> b THEN
   8    dbms_output.put_line(a||'>'||b);
   9   ELSE
  10    dbms_output.put_line(a||'<'||b);
  11   END IF;
  12  END;
  13  /
1< 2
PL/SQL 过程已成功完成。
```

2) 形式二

IF…END IF 语句一次只能判断一个条件,语句 IF…ELSIF…END IF 则可以判定两个以上的判断条件。该语句的语法形式如下。

```
IF 条件 1 THEN
  PL/SQL 语句 1 或 SQL 语句 1;
ELSIF 条件 2 THEN
  PL/SQL 语句 2 或 SQL 语句 2;
…
ELSE
  PL/SQL 语句 n 或 SQL 语句 n;
END IF;
```

【例 3-6】 IF 语句示例 2:根据成绩输出对应的成绩级别。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2   score_var number;
  3  BEGIN
  4   score_var:= 88;
  5   IF score_var< 60 THEN
  6   dbms_output.put_line('差');
  7   ELSIF score_var< 80 THEN
  8    dbms_output.put_line('中');
```

```
  9    ELSIF score_var < 90 THEN
 10     dbms_output.put_line('良');
 11    ELSE
 12     dbms_output.put_line('优');
 13    END IF;
 14   END;
 15   /
良
PL/SQL 过程已成功完成。
```

2. CASE 语句

CASE 语句的作用与 IF…ELSIF…END IF 语句相同,都可以实现多项选择。但 CASE 语句是一种更简洁的表示法,并且相对于 IF 结构表示法而言消除了一些重复。CASE 语句共有两种形式。

1) 形式一

第一种形式是获取一个表达式的值,系统根据其值,查找与其相匹配的 WHEN 常量。当找到一个匹配时,就执行与该 WHEN 常量相关的 THEN 子句;如果没有与表达式相匹配的 WHEN 常量,那么就执行 ELSE 子句。该语句的语法形式如下:

```
CASE 表达式
  WHEN  常量 1  THEN  PL/SQL 语句 1;
  WHEN  常量 2  THEN  PL/SQL 语句 2;
  …
  WHEN  常量 n  THEN  PL/SQL 语句 n;
[ ELSE  PL/SQL 语句 n + 1; ]
END CASE;
```

【例 3-7】 CASE 语句示例 1:判断 emp 表中"SMITH"员工的职务。

```
SQL > SET SERVEROUTPUT ON
SQL > DECLARE
  2   job_var emp.job%type;
  3  BEGIN
  4   SELECT job
  5   INTO job_var
  6   FROM emp
  7   WHERE ename = 'SMITH';
  8   CASE job_var
  9    WHEN 'SALESMAN' THEN dbms_output.put_line('SMITH'||'是销售员');
 10    WHEN 'CLERK' THEN dbms_output.put_line('SMITH'||'是管理员');
 11    ELSE dbms_output.put_line('SMITH'||'是经理');
 12   END CASE;
 13  END;
 14  /
SMITH 是管理员
PL/SQL 过程已成功完成。
```

2) 形式二

第二种形式是判断每个 WHEN 子句后的条件。该语句的语法形式如下:

```
CASE
  WHEN 条件 1 THEN PL/SQL 语句 1;
  WHEN 条件 2 THEN PL/SQL 语句 2;
```

```
  …
  WHEN 条件 n THEN PL/SQL 语句 n;
  [ELSE PL/SQL 语句 n+1;]
END CASE;
```

【例 3-8】 CASE 语句示例 2：假设所给的数值是一个分数，判断该分数的等级。

等级判断标准如下：分数<60 为“差”，60 <=分数<80 为“中”，80 <=分数<90 为“良”，90 <=分数<=100 为“优”。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2   score_var number;
  3  BEGIN
  4   score_var:=85;
  5   CASE
  6     WHEN score_var<60 THEN dbms_output.put_line('差');
  7     WHEN score_var<80 THEN dbms_output.put_line('中');
  8     WHEN score_var<90 THEN dbms_output.put_line('良');
  9     ELSE dbms_output.put_line('优');
 10   END CASE;
 11  END;
 12  /
良
PL/SQL 过程已成功完成。
```

3.1.4 循环语句

循环语句与条件语句一样都能控制程序的执行流程，它允许重复执行一条语句或一组语句。PL/SQL 支持 3 种类型的循环：无条件循环、WHILE 循环和 FOR 循环。

1. 无条件循环

最基本的循环称为无条件循环。这种类型的循环如果没有指定 EXIT 语句将一直进行下去，成为死循环。所以，无条件循环中必须指定 EXIT 语句何时停止执行循环。该语句的语法形式如下：

```
LOOP
  PL/SQL 语句;
  EXIT WHEN 条件;
END LOOP;
```

为了能让循环正常运行，必须为 EXIT WHEN 子句提供一个在某时刻可以判断为 TRUE 的条件。当判断条件为 TRUE 时，就停止循环的执行。

【例 3-9】 LOOP 循环语句示例：求 1～5 的和。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2    i number:=1;
  3    s number:=0;
  4  BEGIN
  5   LOOP
  6    s:=s+i;
  7    i:=i+1;
  8    EXIT WHEN i>5;
```

```
  9    END LOOP;
 10    dbms_output.put_line('1 + 2 + … + 5 = '||s);
 11  END;
 12  /
1 + 2 + … + 5 = 15
PL/SQL 过程已成功完成。
```

2. WHILE 循环

WHILE 循环在每次执行时,都将判断循环条件。如果它为 TRUE,那么循环将继续执行;如果条件为 FALSE,则循环将会停止执行。该语句的语法形式如下:

```
WHILE 条件 LOOP
  PL/SQL 语句;
END LOOP;
```

【例 3-10】 WHILE 循环语句示例:求 1～5 的和。

```
SQL > SET SERVEROUTPUT ON
SQL > DECLARE
  2  i number: = 1;
  3  s number: = 0;
  4 BEGIN
  5  WHILE i < = 5 LOOP
  6   s: = s + i;
  7   i: = i + 1;
  8  END LOOP;
  9  dbms_output.put_line('1 + 2 + … + 5 = '||s);
 10 END;
 11  /
1 + 2 + … + 5 = 15
PL/SQL 过程已成功完成。
```

3. FOR 循环

在 WHILE 循环中,为了防止出现死循环,需要在循环内不断修改判断条件。而 FOR 循环则通过指定一个数字范围,确切地指出循环应执行多少次。该语句的语法形式如下:

```
FOR 循环控制变量 IN [REVERSE] 下限值..上限值 LOOP
  PL/SQL 语句;
END LOOP;
```

使用 FOR 循环时应注意以下两点。

(1) FOR 循环中的下限值和上限值决定了循环的运行次数。默认情况下,循环控制变量从下限值开始。每运行一次,循环计数器的值就会自动加 1;当循环控制变量达到上限值时,FOR 循环结束。

(2) 使用关键字 REVERSE 时,循环控制变量将自动减 1,并强制循环控制变量的值从上限值到下限值。

【例 3-11】 FOR 循环语句示例:计算 1～5 的和。

```
SQL > SET SERVEROUTPUT ON
SQL > DECLARE
  2   s number: = 0;
  3  BEGIN
  4   FOR i IN 1..5 LOOP
```

```
  5     s:=s+i;
  6    END LOOP;
  7    dbms_output.put_line('1+2+…+5='||s);
  8   END;
  9   /
1+2+…+5=15
PL/SQL 过程已成功完成。
```

3.2 游　　标

通过 SELECT 语句查询时，返回的结果是一个由多行记录组成的集合。而程序设计语言有时要处理查询结果集中的每一条记录。为此，SQL 提供了游标机制。游标充当指针的作用，使程序设计语言一次只能处理查询结果中的一行。

在 Oracle 中，有显式和隐式两种游标。对于 PL/SQL 程序中发出的所有 DML（数据操纵语言）和 SELECT 语句，Oracle 都会自动声明“隐式游标”。为了处理由 SELECT 语句返回的一组记录，需要在 PL/SQL 程序中声明和处理“显式游标”。这里主要介绍显式游标的应用。

3.2.1 显式游标的定义和使用

显式游标是在 PL/SQL 程序中使用包含 SELECT 语句来声明的游标。如果需要处理从数据库中检索的一组记录，则可以使用显式游标。使用显式游标处理数据需要 4 个 PL/SQL 步骤：声明游标、打开游标、提取数据和关闭游标。

1. 声明游标

在 DECLARE 部分按以下格式声明游标：

```
CURSOR  游标名  IS  SELECT 语句;
```

声明游标时需注意以下两点：

（1）声明游标的作用是得到一个 SELECT 查询结果集。该结果集中包含了应用程序中要处理的数据，从而为用户提供逐行处理的途径。

（2）SELECT 语句是对表或视图的查询语句。可以带 WHERE 条件、ORDER BY 或 GROUP BY 等子句，但不能使用 INTO 子句。

2. 打开游标

在 BEGIN…END 部分，按以下格式打开游标：

```
OPEN  游标名;
```

游标必须先声明后打开。打开游标时，SELECT 语句的查询结果就被传送到了游标工作区，以便供用户读取。

3. 提取数据

在 BEGIN…END 部分，按以下格式将游标工作区中的数据读取到变量中（提取游标必须在打开游标之后进行）：

```
FETCH  游标名  INTO  变量名 1[,变量名 2,…];
```

成功打开游标后，游标指针指向结果集的第一行之前，而 FETCH 语句将使游标指针指

向下一行。因此,第一次执行 FETCH 语句时,将检索第一行中的数据并保存到变量中。随后每执行一条 FETCH 语句,该指针将移动到结果集的下一行。可以在循环中使用 FETCH 语句,这样每一次循环都会从表中读取一行数据,然后进行相同的逻辑处理。

4. 关闭游标

显式游标打开后,必须显式地关闭。按以下格式关闭游标:

```
CLOSE  游标名;
```

游标一旦关闭,其占用的资源就被释放,用户不能再从结果集中检索数据。如果想重新检索,必须重新打开游标才能使用。

【例 3-12】 用游标提取 emp 表中 7788 员工的姓名和职务。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2   v_ename emp.ename%type;
  3   v_job   emp.job%type;
  4   CURSOR emp_cursor IS SELECT ename,job FROM emp WHERE empno = 7788;
  5  BEGIN
  6   OPEN emp_cursor;
  7   FETCH emp_cursor INTO v_ename,v_job;
  8   dbms_output.put_line(v_ename||''||v_job);
  9   CLOSE emp_cursor;
 10  END;
 11  /
SCOTT ANALYST
PL/SQL 过程已成功完成。
```

【例 3-13】 用游标显示工资最高的前 3 名员工的姓名和工资。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2   v_ename emp.ename%type;
  3   v_sal emp.sal%type;
  4   CURSOR emp_cursor
  5   IS SELECT ename,sal FROM emp ORDER BY sal DESC;
  6  BEGIN
  7   OPEN  emp_cursor;
  8   FOR i IN 1..3 LOOP
  9    FETCH  emp_cursor  INTO  v_ename,v_sal;
 10    dbms_output.put_line(v_ename||''||v_sal);
 11   END LOOP;
 12   CLOSE  emp_cursor;
 13  END;
 14  /
KING  5000
FORD  3000
SCOTT 3000
PL/SQL 过程已成功完成。
```

3.2.2 显式游标的属性

虽然可以使用前面的形式获得游标数据,但是在游标定义以后使用它的一些属性来进行结构控制是一种更为灵活的方法。显式游标的属性如表 3-1 所示。

表 3-1 显式游标的属性

属 性	返回值类型	功 能
%ROWCOUNT	整型	获得 FETCH 语句返回的数据行数
%FOUND	布尔型	FETCH 语句是否提取一行数据，提取成功则为 TRUE，否则为 FALSE
%NOTFOUND	布尔型	与%FOUND 属性的返回值相反
%ISOPEN	布尔型	游标是否已经打开，打开为 TRUE，否则为 FALSE

如果要取得游标属性，在属性前加游标名即可。

【例 3-14】 使用游标显示 dept 表中的每行记录。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2  row_dept dept%rowtype;
  3  CURSOR dept_cursor IS SELECT * FROM dept;
  4  BEGIN
  5  OPEN dept_cursor;
  6  IF dept_cursor%ISOPEN THEN
  7   LOOP
  8    FETCH dept_cursor INTO row_dept;
  9    EXIT WHEN dept_cursor%NOTFOUND;
 10    dbms_output.put_line(row_dept.deptno||''||row_dept.dname||''||row_dept.loc);
 11   END LOOP;
 12  ELSE
 13   dbms_output.put_line('用户信息:游标没有打开!');
 14  END IF;
 15  CLOSE dept_cursor;
 16 END;
 17 /
10 ACCOUNTING NEW YORK
20 RESEARCH DALLAS
30 SALES CHICAGO
40 OPERATIONS BOSTON
PL/SQL 过程已成功完成。
```

3.2.3 游标的 FOR 循环

在 PL/SQL 中还有一种更加方便地使用显式游标的方法，那就是游标的 FOR 循环。游标的 FOR 循环是显式游标的一种快捷使用方式，它使用 FOR 循环依次读取结果集中的行数据。当 FOR 循环开始时，游标将自动打开(不需要使用 OPEN 方法)。每循环一次，系统将自动读取游标当前行的数据(不需要使用 FETCH)；当退出 FOR 循环时，游标被自动关闭(不需要使用 CLOSE)。

语句的定义格式如下：

```
FOR  记录变量名  IN  游标名  LOOP
  PL/SQL 语句;
END LOOP;
```

【例 3-15】 使用 FOR 循环形式显示 10 部门员工的编号和姓名。

```
SQL> SET SERVEROUTPUT ON
```

```
SQL > DECLARE
  2   CURSOR emp_cursor IS SELECT empno,ename FROM emp WHERE deptno = 10;
  3  BEGIN
  4   FOR emp_r IN emp_cursor LOOP
  5    dbms_output.put_line(emp_r.empno||' '||emp_r.ename);
  6   END LOOP;
  7  END;
  8  /
7782 CLARK
7839 KING
7934 MILLER
PL/SQL 过程已成功完成。
```

3.2.4 带参数的游标

在声明游标时,可以将参数传递给游标并在查询中使用。带参数游标的声明语句格式如下:

```
CURSOR 游标名 (参数[,参数,…])
  IS SELECT 语句;
```

其中,参数的定义格式如下:

```
参数名  [IN]  数据类型[:= 值 或 DEFAULT 值]
```

对于参数,需要注意以下两点。

(1) 参数只定义数据类型,没有长度。

(2) DEFAULT 用于给参数设定一个默认值。当没有参数值给游标时,就使用默认值。

打开游标时,可以指定传递的参数值。带参数游标的打开语句格式如下:

```
OPEN 游标名(值[,值,…])
```

【例 3-16】 根据所给的参数值,显示员工编号和姓名。

```
SQL > SET SERVEROUTPUT ON
SQL > DECLARE
  2  v_empno emp.empno % type;
  3  v_ename emp.ename % type;
  4  CURSOR emp_cursor(p_deptno number,p_job varchar2)
  5   IS SELECT empno,ename FROM emp WHERE deptno = p_deptno and job = p_job;
  6  BEGIN
  7   OPEN emp_cursor(10, 'CLERK');
  8   LOOP
  9    FETCH emp_cursor INTO v_empno,v_ename;
 10    EXIT WHEN emp_cursor % NOTFOUND;
 11    dbms_output.put_line(v_empno||' '||v_ename);
 12   END LOOP;
 13   CLOSE emp_cursor;
 14  END;
 15  /
7934 MILLER
PL/SQL 过程已成功完成。
```

3.2.5 使用游标更新和删除数据

通过游标可以查询数据表中的数据,那么如何使用游标修改或删除数据表中的记录呢?

使用游标修改和删除表中记录的操作是指在游标定位后，修改或删除表中指定的数据行。

为了使用游标更新和删除数据，需要在声明游标时使用 FOR UPDATE 选项，以便在打开游标时锁定游标结果集与表中对应数据行的所有列和部分列。使用 FOR UPDATE 选项声明游标的语法格式如下：

```
CURSOR 游标名
  IS SELECT 语句 FOR UPDATE [OF 列 1[,列 2,…]];
```

其中，OF 选项只在要进行数据更新(UPDATE)时使用。OF 后面指定要更新的具体数据列。如果不指定 OF 选项，则可更新游标当前行中的所有数据。

当使用 FOR UPDATE 声明游标后，可在 DELETE 和 UPDATE 语句中使用 WHERE CURRENT OF 子句，修改或删除游标结果集中当前行对应的表中的数据行。格式如下：

```
WHERE CURRENT OF 游标名
```

【例 3-17】 使用游标更新 emp 表中的 COMM 值。

代码如下：

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2   CURSOR c1 IS SELECT empno,sal FROM emp
  3    WHERE comm IS NULL FOR UPDATE OF comm;
  4   v_comm emp.sal%TYPE;
  5  BEGIN
  6   FOR r IN c1 LOOP
  7    CASE
  8     WHEN r.sal < 500 THEN v_comm := r.sal * 0.25;
  9     WHEN r.sal < 1000 THEN v_comm := r.sal * 0.2;
 10     WHEN r.sal < 3000 THEN v_comm := r.sal * 0.15;
 11     ELSE v_comm := r.sal * 0.12;
 12    END CASE;
 13    UPDATE emp SET comm = v_comm WHERE CURRENT OF c1;
 14   END LOOP;
 15  END;
 16  /
PL/SQL 过程已成功完成。
```

3.3 异常处理

异常是 Oracle 数据库中的 PL/SQL 代码执行期间出现的错误。发生异常后，语句将停止执行，跳转到 PL/SQL 程序块的异常处理部分。SQL Plus 处理异常的方法就是在屏幕上显示异常信息。

Oracle 常用的有两种类型的异常。

(1) 预定义异常。Oracle 为用户提供了大量的在 PL/SQL 中使用的预定义异常，以检查用户代码失败的一般原因。

(2) 自定义异常。如果程序设计人员认为某种情况违反了业务逻辑，设计人员可明确定义并触发异常。

异常处理部分一般放在 PL/SQL 程序块的后半部分，其结构如下：

```
EXCEPTION
  WHEN  异常情况 1  THEN  处理异常代码 1;
  WHEN  异常情况 2  THEN  处理异常代码 2;
  …
   WHEN  OTHERS  THEN  处理异常代码;
```

3.3.1 Oracle 的预定义异常

对于 Oracle 提供的预定义异常,用户可以在自己的 PL/SQL 异常处理部分使用名称对其进行标识。常用的预定义异常及其对应的 Oracle 错误信息如表 3-2 所示。

表 3-2 Oracle 的预定义异常

错 误 信 息	异 常 名 称	说 明
ORA-0001	DUP_VAL_ON_INDEX	试图破坏一个唯一性限制
ORA-0051	Timeout-on-resource	在等待资源时发生超时
ORA-0061	Transaction-backed-out	由于发生死锁,事务被撤销
ORA-1001	Invalid-CURSOR	试图使用一个无效的游标
ORA-1012	Not-logged-on	没有连接到 Oracle
ORA-1017	Login-denied	无效的用户名/口令
ORA-1403	NO_DATA_FOUND	SELECT INTO 没有找到数据
ORA-1422	TOO_MANY_ROWS	SELECT INTO 返回多行
ORA-1476	Zero-divide	试图被零除
ORA-1722	Invalid-NUMBER	转换一个数字时失败
ORA-6500	Storage-error	内存不够引发的内部错误
ORA-6501	Program-error	内部错误
ORA-6502	Value-error	转换或截断错误
ORA-6504	Rowtype-mismatch	主变量和游标的类型不兼容
ORA-6511	CURSOR-ALREADY-OPEN	试图打开一个已经打开的游标时,将产生这种异常
ORA-6530	Access-INTO-null	试图为 null 对象的属性赋值

【例 3-18】 向 dept 表中插入与主键值相同的记录。

情况一 不用预定义异常解决,代码如下:

```
SQL > BEGIN
  2    INSERT INTO dept(deptno,dname) VALUES(10,'HR');
  3  END;
  4  /
BEGIN
 *
第 1 行出现错误:
ORA - 00001: 违反唯一约束条件 (SCOTT.PK_DEPT)
ORA - 06512: 在 line 2
```

情况二 使用预定义异常解决,代码如下:

```
SQL > SET SERVEROUTPUT ON
SQL > BEGIN
  2    INSERT INTO dept(deptno,dname) VALUES(10,'HR');
  3  EXCEPTION
  4    WHEN DUP_VAL_ON_INDEX THEN
```

```
  5    dbms_output.put_line('捕获到了 DUP_VAL_ON_INDEX 异常');
  6    dbms_output.put_line('该主键值已经存在');
  7  END;
  8  /
捕获到了 DUP_VAL_ON_INDEX 异常
该主键值已经存在

PL/SQL 过程已成功完成。
```

3.3.2 用户自定义异常的处理

在实际的程序开发中，为了实施具体的业务逻辑规则，程序开发人员往往会根据这些逻辑规则自定义一些异常。当用户违反了这些规则时，就会引发一个自定义异常，从而中断程序的正常执行，并转到自定义异常处理部分。

用户自定义异常是通过显式使用 RAISE 语句来触发的。当引发一个异常时，控制就转向 EXCEPTION 异常处理部分，执行异常处理语句。处理自定义异常的步骤如下。

（1）在 PL/SQL 程序块的定义部分定义异常情况，语句如下：

```
异常情况  EXCEPTION;
```

（2）使用 RAISE 引出自定义异常，语句如下：

```
RAISE  异常情况;
```

（3）在 PL/SQL 程序块的异常处理部分对异常情况做出相应的处理。

【例 3-19】 检查 dept 表中的记录是否被更新。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2   ex_update EXCEPTION;
  3  BEGIN
  4   UPDATE dept SET dname = 'HR' WHERE deptno = 50;
  5   IF sql % notfound THEN
  6     RAISE ex_update;
  7   END IF;
  8  EXCEPTION
  9   WHEN ex_update THEN
 10    dbms_output.put_line('捕获到自定义异常 ex_update');
 11    dbms_output.put_line('未更新任意行');
 12  END;
 13  /
捕获到自定义异常 ex_update
未更新任意行

PL/SQL 过程已成功完成。
```

3.4 存储过程

前面所创建的 PL/SQL 程序块都是匿名的。这些匿名的程序块没有被存储，每次执行后都不可以被重新使用。因此，每次运行匿名程序块时，都需要先编译然后再执行。很多时候都需要保存 PL/SQL 程序块，便于以后可以重新使用。

存储过程是一种命名的 PL/SQL 程序块,它可以被赋予参数并存储在数据库中,以便被用户调用。由于存储过程是已经编译好的代码,所以在调用的时候不必再次进行编译,从而提高了程序的运行效率。

3.4.1 创建存储过程

创建存储过程的语法结构如下:

```
CREATE [OR REPLACE] PROCEDURE 过程名 AS
  声明部分;
BEGIN
  功能语句;
EXCEPTION
  异常处理;
END;
```

说明:选择 OR REPLACE 选项时,如果创建的存储过程已经存在,将重新建立与原来同名的存储过程。

【例 3-20】 存储过程示例。

```
SQL > CREATE OR REPLACE PROCEDURE emp_P AS
  2   emp_row emp % ROWTYPE;
  3  BEGIN
  4   SELECT * INTO emp_row FROM emp WHERE job = 'CLERK';
  5   dbms_output.put_line(emp_row.empno||''||emp_row.ename);
  6  EXCEPTION
  7   WHEN TOO_MANY_ROWS THEN
  8    dbms_output.put_line('捕获到了 TOO_MANY_ROWS 异常');
  9    dbms_output.put_line('SELECT 语句检索到了多行数据');
 10  END;
 11  /

过程已创建。
```

3.4.2 调用存储过程

存储过程创建后,就可以任意调用该过程了。

可以使用 EXECUTE 语句直接调用存储过程。EXECUTE 语句的语法形式如下:

```
EXEC[UTE]  过程名;
```

【例 3-21】 调用执行例 3-20 所创建的存储过程。

```
SQL > SET SERVEROUTPUT ON
SQL > EXEC emp_P;
捕获到了 TOO_MANY_ROWS 异常
SELECT 语句检索到了多行数据

PL/SQL 过程已成功完成。
```

3.4.3 存储过程的参数

在创建存储过程时,需要考虑存储过程的灵活应用,以便重新使用它们。通过使用“参

数”可以使程序单元变得灵活。参数是一种向程序单元输入/输出数据的机制。存储过程可以接收和返回 0 到多个参数。Oracle 有 3 种参数模式：IN、OUT 和 IN OUT。

带参数存储过程的创建语法格式如下：

```
CREATE [OR REPLACE] PROCEDURE 过程名(
  参数 1 [ IN | OUT | IN OUT] 数据类型,
  参数 2 [ IN | OUT | IN OUT] 数据类型,
  …
) AS
  声明部分;
BEGIN
  功能语句;
EXCEPTION
  异常处理;
END;
```

1. IN 参数

IN 参数为输入参数。该参数值由调用者传入，并且只能被存储过程读取。

【例 3-22】 创建一个向 dept 表中插入新记录的存储过程 dept_p。

```
SQL> CREATE OR REPLACE PROCEDURE dept_p(
  2  p_deptno IN number,
  3  p_dname IN varchar2,
  4  p_loc IN varchar2
  5  ) AS
  6  BEGIN
  7   INSERT INTO dept VALUES(p_deptno,p_dname,p_loc);
  8  EXCEPTION
  9   WHEN DUP_VAL_ON_INDEX THEN
 10    dbms_output.put_line('重复的部门编号');
 11  END;
 12  /

过程已创建。
```

打开系统默认设置的输出功能，代码如下：

```
SQL> SET SERVEROUTPUT ON
SQL> EXEC dept_p(50,'HR','CHINA');

PL/SQL 过程已成功完成。
```

查看修改后的表，代码如下：

```
SQL> SELECT * FROM dept WHERE deptno = 50;
```

语句执行结果为：

```
    DEPTNO  DNAME         LOC
---------  ---------  ---------
        50  HR            CHINA
```

2. OUT 参数

OUT 参数为输出参数，该类型的参数值由存储过程写入。OUT 类型的参数适用于存储过程向调用者返回一个或多个数据的情况。

【例 3-23】 创建存储过程 dept_p,该过程根据提供的部门编号返回部门的名称和地址。

```
SQL> CREATE OR REPLACE PROCEDURE dept_p(
  2  i_no IN dept.deptno%TYPE,
  3  o_name OUT dept.dname%TYPE,
  4  o_loc OUT dept.loc%TYPE
  5  ) AS
  6  BEGIN
  7   SELECT dname,loc INTO o_name,o_loc FROM dept WHERE deptno = i_no;
  8  EXCEPTION
  9   WHEN NO_DATA_FOUND THEN
 10    o_name:= 'NULL';
 11    o_loc:= 'NULL';
 12  END;
 13  /

过程已创建。
```

因为该存储过程要通过 OUT 参数返回值,这意味着在调用它时必须提供能够接收返回值的变量。因此,在调用前需要使用 VARIABLE 命令定义变量接收返回值,并且在调用存储过程时需要在变量前加冒号。

【例 3-24】 调用例 3-23 创建的存储过程,输出指定部门编号的部门名称和地址。

使用 VARIABLE 命令定义变量接收返回值,代码如下:

```
SQL> VARIABLE v_dname varchar2(20);
SQL> VARIABLE v_loc varchar2(10);
SQL> EXEC dept_p(10,:v_dname,:v_loc);

PL/SQL 过程已成功完成。
```

输出部门名称和地址,代码如下:

```
SQL> PRINT v_dname;

V_DNAME
---------------
ACCOUNTING

SQL> PRINT v_loc;

V_LOC
---------------
NEW YORK
```

3. IN OUT 参数

IN 参数可以接收一个值,但是不能在过程中修改这个值。而对于 OUT 参数而言,它在调用过程时为空,在过程的执行中将为这个参数指定一个值,并在执行结束后返回。而 IN OUT 类型的参数同时具有 IN 参数和 OUT 参数的特性,在过程中可以读取和写入该类型参数。

【例 3-25】 使用 IN OUT 参数实现两个数的交换。

创建存储过程的代码如下:

```
SQL> CREATE OR REPLACE PROCEDURE swap(
```

```
  2  p_num1 IN OUT number,
  3  p_num2 IN OUT number
  4  ) AS
  5  var_temp number;
  6  BEGIN
  7   var_temp: = p_num1;
  8   p_num1: = p_num2;
  9   p_num2: = var_temp;
 10  END;
 11  /
```

过程已创建。

实现两个数的交换,代码如下:

```
SQL > SET SERVEROUTPUT ON
SQL > DECLARE
  2   var_max number: = 10;
  3   var_min number: = 18;
  4  BEGIN
  5   IF var_max < var_min THEN
  6    swap(var_max,var_min);
  7   END IF;
  8   dbms_output.put_line('var_max = '||var_max);
  9   dbms_output.put_line('var_min = '||var_min);
 10  END;
 11  /
var_max = 18
var_min = 10

PL/SQL 过程已成功完成。
```

3.4.4 修改/删除存储过程

在 Oracle 中,如果要修改存储过程,应使用 CREATE OR REPLACE PROCEDURE 语句,也就是覆盖原有的存储过程。

【例 3-26】 修改例 3-25 创建的存储过程 swap。

创建存储过程的代码如下:

```
SQL > CREATE OR REPLACE PROCEDURE swap AS
  2  BEGIN
  3   dbms_output.put_line('这是修改后的存储过程');
  4  END;
  5  /
```

过程已创建。

查看修改后的存储过程,代码如下:

```
SQL > EXEC swap;
这是修改后的存储过程

PL/SQL 过程已成功完成。
```

删除存储过程可以使用 DROP 语句,其语法格式如下:

```
DROP PROCEDURE 存储过程名;
```

【例 3-27】 删除例 3-25 创建的存储过程 swap。

代码如下：

```
SQL> DROP PROCEDURE swap;

过程已删除。
```

3.4.5 查看存储过程的错误

编写的存储过程难免会出现各种错误而导致编译失败。为了缩小排查错误的范围，Oracle 提供了查看存储过程错误的语句，其语法格式如下：

```
SHOW ERRORS PROCEDURE 存储过程名;
```

【例 3-28】 创建一个有错误的存储过程，然后查看错误信息。

代码如下：

```
SQL> CREATE OR REPLACE PROCEDURE test_proc
  2  AS
  3  BEGIN
  4   dbmm_output.put_line('这是有错误的存储过程,将 dbms 写成了 dbmm');
  5  END;
  6  /
```

警告：创建的过程带有编译错误。

查看错误的具体细节，代码如下：

```
SQL> SHOW ERRORS PROCEDURE test_proc;
PROCEDURE TEST_PROC 出现错误:

LINE/COL    ERROR
--------  ---------
4/2         PL/SQL: Statement ignored
4/2         PLS-00201: 必须声明标识符 'DBMM_OUTPUT.PUT_LINE'
```

从错误提示可知，错误是由第 4 行引发的，正确的写法如下：

```
dbms_output.put_line('这是有错误的存储过程,将 dbms 写成了 dbmm');
```

3.5 小　　结

本章介绍了 PL/SQL 程序块定义部分、执行部分和异常处理部分的作用以及编写方法。注意：编写 PL/SQL 程序块时，执行部分是必需的，而定义部分和异常处理部分是可选的。

PL/SQL 程序块中的 IF 语句可执行简单条件判断、二重分支判断和多重分支判断。当使用 IF 语句时，注意 END IF 是两个词，而 ELSIF 是一个词。CASE 语句可执行多重分支判断。WHILE 语句和 FOR 语句执行循环控制操作的方法。

在使用 SELECT 语句查询数据库时，查询返回的数据存放在结果集中。用户在得到结果集后，需要逐行逐列地获取其中存储的数据，以便在应用程序中使用这些值。游标机制可完成此类操作。如果使用游标更新或删除数据，则在定义游标时必须指定 FOR UPDATE

子句，当更新或删除游标行时必须带有 WHERE CURRENT OF 子句。

当 PL/SQL 运行错误时，可以使用预定义异常和用户自定义异常的方法来处理发生的错误。

存储过程是用于执行特定操作的 PL/SQL 程序块，在需要时可以直接调用，提高代码的重用性和共享性。

SQL 的用户可以是终端用户，也可以是应用程序。嵌入式 SQL 将 SQL 作为一种数据子语言嵌入高级语言中，利用高级语言和其他专门软件来弥补 SQL 语句在实现复杂应用方面的不足。动态 SQL 允许在执行一个应用程序时，根据不同的情况动态地定义和执行某些 SQL 语句。动态 SQL 可实现应用中的灵活性。SQL 已经成为数据库的主流语言，其意义也远远超过数据库范围。

习 题 三

一、选择题

1. 下列哪条语句允许检查 UPDATE 语句所影响的行数？（ ）

A. SQL%FOUND　　B. SQL%ROWCOUNT

C. SQL%COUNT　　D. SQL%NOTFOUND

2. 在定义游标时，使用 FOR UPDATE 子句的作用是（ ）。

A. 执行游标　　B. 执行 SQL 语句的 UPDATE 语句

C. 对要更新表的列进行加锁　　D. 以上都不对

3. 对于游标 FOR 循环，以下哪一种说法是不正确的？（ ）

A. 循环隐含使用 FETCH 获取数据　　B. 循环隐含使用 OPEN 打开记录集

C. 终止循环操作也就关闭了游标　　D. 游标 FOR 循环不需要定义游标

4. 下列哪个关键字用来在 IF 语句中检查多个条件？（ ）

A. ELSE IF　　B. ELSEIF　　C. ELSIF　　D. ELSIFS

5. 如何终止 LOOP 循环，而不会出现死循环？（ ）

A. 当 LOOP 语句中的条件为 FALSE 时停止

B. 这种循环限定的循环次数会自动终止循环

C. EXIT WHEN 语句中的条件为 TRUE

D. EXIT WHEN 语句中的条件为 FALSE

6. 如果 PL/SQL 程序块的可执行部分引发了一个错误，则程序的执行顺序将发生什么变化？（ ）

A. 程序将转到 EXCEPTION 部分运行

B. 程序将中止运行

C. 程序仍然正常运行

D. 以上都不对

二、填空题

1. PL/SQL 程序块主要包含 3 个主要部分：声明部分、可执行部分和_____部分。

2. 使用显式游标主要有 4 个步骤：声明游标、_____、检索数据、_____。

3. 在 PL/SQL 中,如果 SELECT 语句没有返回列,则会引发 Oracle 错误,并引发_____异常。

4. 查看下面的程序块,其中变量 var_b 的结果为_____:

```
DECLARE
  var_a number: = 1200;
  var_b number;
 BEGIN
  IF var_a > 500 THEN
    var_b: = 5;
  ELSIF var_a > 1000 THEN
    var_b: = 10;
  ELSE
    var_b: = 8;
  END IF;
END;
```