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 语言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 10g 中与数据库编程相关的内容。

3.1 PL/SQL 编程基础

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 程序通常包括三部分：

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 和 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，其中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 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，其中 BEGIN 和 END 部分是 PL/SQL 必需的部分，但一般的程序块都包括这三部分，其基本结构如图 3-1 所示。

定义部分
DECLARE
创建变量、常量、游标、异常等
执行部分
BEGIN
SQL 语句、PL/SQL 的流程控制语句
EXCEPTION
异常处理代码
END;

图 3-1 PL/SQL 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( * )
  5    INTO sum_num
  6    FROM dept;
  7    dbms_output.put_line('记录个数: '||sum_num);
  8  END;
  9  /
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，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
  9   WHERE deptno = 10;
 10   dbms_output.put_line(no||' '||name||' '||place);
 11  END;
 12  /
10 ACCOUNTING NEW YORK

PL/SQL 过程已成功完成。
```

使用%TYPE 定义变量的好处：

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，如果用户随后修改数据库结构（如改变某列的数据类型），则用户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
  7   WHERE deptno = 10;
  8   dbms_output.put_line(row_dept.deptno);
  9   dbms_output.put_line(row_dept.dname||' '||row_dept.loc);
 10  END;
 11  /
10
ACCOUNTING NEW YORK
```

```
PL/SQL 过程已成功完成。
```

3.1.3 条件判断语句

PL/SQL 与其他的编程语言一样,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。在 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,而语句 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。判断某一年是否为闰年。

闰年的判断条件为：年号能被 4 整除但不能被 100 整除，或者能被 400 整除。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2   year_data number;
  3   leap       Boolean;
  4  BEGIN
  5   year_data := 2012;
  6   IF mod(year_data,4)<> 0 THEN
  7     leap := false;
  8   ELSIF mod(year_data,100)<> 0 THEN
  9     leap := true;
 10   ELSIF mod(year_data,400)<> 0 THEN
 11     leap := false;
 12   ELSE
 13     leap := true;
 14   END IF;
 15   IF leap THEN
 16    DBMS_OUTPUT.PUT_LINE(year_data|| '是闰年');
 17   ELSE
 18    DBMS_OUTPUT.PUT_LINE(year_data||'是平年');
 19   END IF;
 20  END;
 21  /
2012 是闰年

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选择器的值，系统根据其值，查找与此相匹配的 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。如果没有与选择器相匹配的 WHEN 常量，那么就执行 ELSE 子句。该语句的语法形式如下。

```
CASE  <条件>
  WHEN  <表达式 1 > THEN PL/SQL 语句 1;
```

```
  WHEN  <表达式 2> THEN PL/SQL 语句 2;
  …
  WHEN  <表达式 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

第二种形式是不使用选择器,而是判断每个 WHEN 子句中的条件。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 >= 60 and score_var < 80 THEN dbms_output.put_line('中');
  8    WHEN score_var >= 80 and 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 支持三种类型的循环：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，循环将一直运行，成为死循环。所以，无条件循环中必须指定 EXIT 语句停止执行循环。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。依次输出 1～5 的平方数。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2   i number := 1;
  3  BEGIN
  4   LOOP
  5    dbms_output.put_line(i||'的平方数为:'||i * i);
  6    i := i + 1;
  7    EXIT WHEN i > 5;
  8   END LOOP;
  9  END;
 10  /
1 的平方数为:1
2 的平方数为:4
3 的平方数为:9
```

```
4 的平方数为:16
5 的平方数为:25

PL/SQL 过程已成功完成。
```

2. WHILE 循环

WHILE 循环在每次执行循环时,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。计算 100 以内能被 3 整除的整数之和。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2    i number := 1;
  3    sum_num number := 0;
  4  BEGIN
  5    WHILE i<100 LOOP
  6     IF mod(i,3) = 0 THEN
  7       sum_num := sum_num + i;
  8     END IF
  9     i := i + 1;
 10   END LOOP;
 11   dbms_output.put_line(sum_num);
 12  END;
 13  /
1683

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,以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;
```

(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。输出 20～1 内能被 3 整除的数据。

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2   FOR i IN REVERSE 1..20 LOOP
  3    IF mod(i,3) = 0 THEN
  4     dbms_output.put_line(i);
  5    END IF;
  6   END LOOP;
  7  END;
  8  /
18
15
12
9
6
3

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。而程序设计语言并不能处理以集合形式返回的数据，为此，SQL 提供了游标机制。游标充当指针的作用，使应用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。对于在 PL/SQL 程序中所有发出的 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四个 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;
```

(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保存到变量中。在随后每执行一个 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,游标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 雇员的姓名和职务。

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2    v_ename emp.ename%type;
  3    v_job emp.job%type;
  4    CURSOR emp_cursor
  5     IS SELECT ename,job FROM emp WHERE empno = 7788;
  6  BEGIN
  7    OPEN emp_cursor;
  8    FETCH emp_cursor INTO v_ename,v_job;
  9    dbms_output.put_line(v_ename||' '||v_job);
 10    CLOSE emp_cursor;
 11  END;
 12  /
SCOTT ANALYST

PL/SQL 过程已成功完成。
```