高级PL/SQL

-集合及游标

单世民

PL/SQL集合

- Oracle的第一个集合就是在Oracle 7中引入的**PL/SQL表**。Oracle 8又添加了两种新的集合类型，同时PL/SQL表也被更名为**index-by表**。Oracle 10g中index-by表又被再次更名，现在它的名字叫**联合数组**。集合使我们可以在表的一行记录中存储多个数据集。

- 集合就是列表，可能有序也可能无序。**所有的集合**（不包括记录）**都是一维的**，可使用标准的数据类型或用户自定义的数据类型（记录或对象）。虽然记录和对象更像是多维集合，但Oracle数据库还是将其看作一维集合。集合类型是在PL/SQL程序设计中经常使用的一些很关键的结构类型。

PL/SQL集合

- 记录
- PL/SQL表（联合数组，index-by 表）
- 嵌套表
- VARRAY

记录

- 记录是在Oracle7引入的，它是有几个相关值构成的复合变量，是表中单行数据结构的一个镜像，可用于支持SELECT语句的返回值。记录将一行数据看成一个单元，而不是将每一列单独处理

记录

- 当在PL/SQL中使用记录时，首先需要定义记录的结构，然后可以设置记录类型的变量。这与通常定义变量的方式不同，因为大多数标量类型或 %type / %rowtype 类型都已经定义了。当用户声明了记录类型的变量之后，就可以为记录变量的单独属性赋值。

记录类型的定义

- 显式定义

```
TYPE record_type IS RECORD
        (
                Variable_1 datatype1[,
                Variable_2 datatype2[,
                ...]]
        );
```

- 隐式定义

  使用%ROWTYPE和%TYPE

记录

- 记录示例

```
DECLARE
    TYPE NewRecordTYPE IS RECORD
    (
           id number,
           name varchar2(10)
    );
    recTest NEWRECORDTYPE;
BEGIN
    SELECT empno,ename INTO recTest
           FROM emp WHERE empno=7369;
    DBMS_OUTPUT.PUT_LINE
           (recTest.id||'号员工的姓名是'||recTest.name);
END;
/
```

VARRAY

- Varray（Varrays）是Oracle 10g数据类型或用户自定义的记录/对象类型的一维结构体。可以在表、记录和对象定义中使用varray，也可以在SQL和PL/SQL中访问它们。可以使用顺序索引值来引用varray中的元素。

VARRAY

- 定义语法

```
TYPE type_name IS {VARRAY|VARING ARRAY}(size)
                  OF data_type [NOT NULL];
```

```
CREATE OR REPLACE
TYPE type_name IS {VARRAY|VARING ARRAY}(size)
                  OF data_type [NOT NULL];
```

VARRAY

- Varray是密集数组，作用更像是传统的程序设计数组，存储在永久性的表中，并可以通过SQL语句进行访问。在创建时，它们都有一个固定的大小，而且这个大小不能改变。Varray中的元素可以使用数字序列的下标进行访问，其**下标索引值从1开始**。
- 默认情况下，Varray中可以出现null值。

VARRAY

- VARRAY示例

```
DECLARE
    TYPE int_varray IS VARRAY(2) OF INTEGER;
    varray_int INT_VARRAY:=int_varray(null,null);
BEGIN
    varray_int(1):=3;
    varray_int(2):=4;
    FOR i IN 1..2 LOOP
       dbms_output.put_line('int_varray['||i||']='||
          varray_int(i));
    END LOOP;
END;
/
```

使用Varray类型数据必须进行初始化！！

VARRAY

- VARRAY与SQL——不得不说的故事
  Varray的强大功能并不局限在过程化的程序设计当中。从Oracle 8到Oracle 10g，Varray提供了一些独特的表现数据的功能。这也是Oracle数据库能成为闻名的对象-关系数据库管理系统（ORDBMS）的原因之一。

VARRAY

- VARRAY与SQL

```
CREATE OR REPLACE TYPE addr_varray
                  AS VARRAY(2) OF VARCHAR2(50 CHAR);
```

```
CREATE TABLE empaddr
    (
       id          number(38) not null,
       name        varchar2(20) not null,
       address     ADDR_VARRAY not null,
       constraint pk_empaddr primary key (id)
    );
```

VARRAY

- VARRAY与SQL

```
INSERT INTO empaddr VALUES(
    1,'Shan Shimin',
    addr_varray('Office 302 of School of Software',
                'Dalian University of Technology')
);
```

```
UPDATE empaddr
SET address=
      addr_varray('Office 303 of School of Software',
                  'Dalian University of Technology')
WHERE ID=1;
```

插入操作是以要么全有要么全无的方式处理Varray类型数据的

VARRAY

- VARRAY与SQL

```
SELECT address FROM empaddr;
```

- 上述查询的方法无法得到什么有用的输出，需要定义一个**嵌套表**集合结构，才能真正访问到varray类型的数据。

VARRAY

- VARRAY与SQL

```
CREATE OR REPLACE TYPE varray_nested_tab
                    IS TABLE OF VARCHAR2(50 CHAR);
```

```
SELECT COLUMN_VALUE FROM THE(
    SELECT CAST(address AS varry_nested_tab)
    FROM empaddr
    WHERE id=1
    );
```

保留字column_value是访问嵌套表中记录行的一种方法（嵌套表中还有一个隐藏列nested_table_id，映射到父表中的记录行上）。
保留字THE用于从嵌套表的一个查询中检索出column_value列。

嵌套表

- 嵌套表是在Oracle 8中引入的集合类型。最开始被定义为密集数组，但是将记录从中删除后，它就变成稀疏数组。它可以存储在永久性的数据表中，可以使用SQL进行访问，也可以进行动态扩展。

- 嵌套表的大小是可以动态分配的。

嵌套表

- 定义语法

```
CREATE OR REPLACE
TYPE type_name AS TABLE OF data_type [NOT NULL];
```

- 嵌套表的使用方式与varray类似，但是两者至少在两方面存在区别：
  - 嵌套表可以动态调整它的大小。
  - 使用嵌套表定义数据列时，此数据列不能采用NOT NULL约束。

PL/SQL表

- PL/SQL表，有时也称为索引表，是可以在PL/SQL例程中使用、能够模仿数组的非永久表。
- 用户可以定义一个PL/SQL表类型，然后声明这种类型的变量，接下来就可以将记录增加到用户的PL/SQL表中，并且采用与引用数组元素大体相同的方式引用这些记录。这些表是一维数组，不要将它们与Oracle表对象混淆。

PL/SQL表

- PL/SQL表示例

```
set serveroutput on
DECLARE
    TYPE my_text_table_type IS table of varchar2(20)
                               INDEX BY binary_integer;
    l_text_table my_text_table_type;

BEGIN
    l_text_table(1) := '一条高速公路';
    l_text_table(2) := '一大片麦田';
    dbms_output.put_line(
    '我们有'||l_text_table.count||'个varchar2变量');
    dbms_output.put_line('变量(1)='||l_text_table(1));
    dbms_output.put_line('变量(2)='||l_text_table(2));
END;
/
```

PL/SQL表

- PL/SQL表不必是密集数组，可以是稀疏数组，其下标值可以是非常离散的值，甚至可以是字符串的哈希值。此时，要遍历PL/SQL表就需要借助于其特定的成员变量（first，last，next）。其中，next类似于Java语言中对于集合类型的迭代器（Iterator）。

PL/SQL表

```
set serveroutput on
DECLARE
    TYPE my_text_table_type IS TABLE OF VARCHAR2(200)
            INDEX BY binary_integer;
    l_text_table my_text_table_type;
    l_index NUMBER;
BEGIN
    FOR emp_rec IN (select * from emp) loop
       l_text_table(emp_rec.empno) := emp_rec.ename;
    END LOOP;
    l_index := l_text_table.first;
    LOOP
       EXIT WHEN l_index IS NULL;
       dbms_output.put_line(l_index||' : '||
                            l_text_table(l_index));
       l_index := l_text_table.next(l_index);
    END LOOP;
END;
/
```

游标

- 游标是一种PL/SQL控制结构，可以对SQL语句的处理进行显式控制，便于对游标的行数据逐条进行处理。

- 游标分类
  - 显式游标
  - 隐式游标

显式游标

- 显式游标是必须通过编写必要的PL/SQL例程来进行管理的游标。游标的整个生命期都在用户的控制之下，因此，用户可以详细地控制PL/SQL怎样在结果集中访问记录。用户可以定义游标、打开游标、从游标中获取数据、使用合适的PL/SQL代码关闭游标。

显式游标

- 定义语法：

```
CURSOR cursor_name [(parameter[, parameter]…)]
    [RETURN return_type] IS select statement;
```

- 显式游标的使用方法：
  1. 定义（**CURSOR** cur_name **IS** ..）
  2. 打开（**OPEN** cur_name）
  3. 取值（**FETCH** cur_name **INTO**..）
  4. 关闭（**CLOSE** cur_name）

遍历显式游标的3种方式

1. 简单LOOP

2. WHILE LOOP

3. FOR LOOP

使用FOR LOOP方式的独特之处在于它不需要显式的OPEN、FETCH或CLOSE命令。
另外，FOR LOOP还使用了一个在代码块的声明部分从没有声明过的一个变量

游标的使用

```
DECLARE
    CURSOR curTest IS
           SELECT * FROM emp;
    recEmp emp%ROWTYPE;
BEGIN
    OPEN curTest;
    FETCH curTest INTO recEmp;
    WHILE curTest%FOUND LOOP
       DBMS_OUTPUT.PUT_LINE(recEmp.empno||
                  '号员工的姓名是'||recEmp.ename);
       FETCH curTest INTO recEmp;
    END LOOP;
    CLOSE curTest;
END;
/
```

游标的属性

%FOUND	检验FETCH语句是否取得了记录
%ISOPEN	检查游标当前是否处在打开状态
%NOTFOUND	%FOUND的相反属性
%ROWCOUNT	检测任意给定的时刻，已从游标中获取的记录行数
%BULK_EXCEPTIONS	为批操作或Bulk Collect操作中产生的异常提供相关信息
%BULK_ROWCOUNT	提供Bulk操作过程中更改的行数信息

游标的使用

```
DECLARE
    CURSOR cur_with_Param(did number) IS
        select * from emp where deptno=did;
    recemp emp%rowtype;
BEGIN
    OPEN cur_with_Param(10);
    LOOP
        IF cur_with_Param%NOTFOUND=true THEN
            dbms_output.put_line('游标的当前NOTFOUND状态为true');
        END IF;
        FETCH cur_with_Param INTO recemp;
        EXIT WHEN cur_with_Param%NOTFOUND;
        dbms_output.put_line(recemp.ename||'是10号部门的员工。');
    END LOOP;
    dbms_output.put_line('10号部门共有员工'||
                                cur_with_Param%ROWCOUNT||'名');
    CLOSE cur_with_Param;
end;
/
```

游标的使用

```
DECLARE
    CURSOR cur_with_param(did number) IS
           SELECT * from emp where deptno=did;
BEGIN
    FOR recEmp IN cur_with_param(10) LOOP
       DBMS_OUTPUT.PUT_LINE(recEmp.ename||
                                  '是10号部门的员工。');
    END LOOP;
END;
/
```

隐式游标

- 可以认为有两种隐式游标
  - 游标子查询
  - 系统隐式游标

系统隐式游标

- 系统隐式游标由Oracle自动打开和关闭。Oracle执行的每一个DML语句都会占用一个内存区域，并以此拥有一个游标
- 系统隐式游标的使用过程不需要开发人员参与交互，不使用OPEN,FETCH和CLOSE命令，但是游标的6个属性还是仍然使用。可以通过使用类似于SQL%NOTFOUND，SQL%ROWCOUNT的方式使用属性。

在系统隐式游标中使用%ISOPEN总是会返回FALSE，因为系统隐式游标总会自动关闭

隐式游标

- SQL

```
BEGIN
    UPDATE emp
    SET sal=sal*0.90
    WHERE deptno=30;

    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT||'行数据被更改');

    IF SQL%NOTFOUND THEN
        DBMS_OUTPUT.PUT_LINE('无法更新10号部门员工的工资');
    END IF;

    COMMIT;
END;
/
```

隐式游标

```
BEGIN
    FOR recEmp IN (select ename
                   from emp where deptno=10)  LOOP

       DBMS_OUTPUT.PUT_LINE(recEmp.ename||
                                    '是10号部门的员工。');
    END LOOP;
END;
/
```

小结

- 集合
  - 记录
  - Varray
  - 嵌套表
  - PL/SQL表
- 游标
  - 显式游标
  - 隐式游标

The End