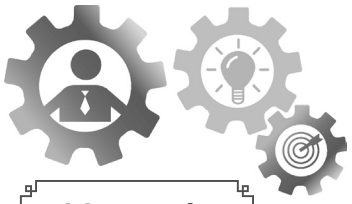

第 3 章

PL/SQL 的复合数据类型

在 PL/SQL 数据类型中，复合数据类型包括集合类型和记录类型。其中，集合类型是多个相同类型的分量的集合，类似于高级语言中的数组。而记录类型是多个不同类型的变量的集合，类似于高级语言中的结构体类型。通常，在 PL/SQL 程序中，集合对应于表中某一列的多个值的集合，而记录类型对应一条记录中若干个字段的集合。本章将主要介绍以下内容。

- 集合类型。
- 记录类型。

3.1 集合类型

在集合类型中，内部元素始终具有相同的数据类型，可以通过集合变量的唯一索引访问它的每个元素。语法格式如下。

```
variable_name(index)
```

创建集合变量，需要先定义集合类型，然后创建该类型的变量，或者使用%TYPE 创建。

PL/SQL 有 3 种集合类型。

(1) 关联数组(Associative Array)。

(2) 可变数组(Varray)。

(3) 嵌套表(Nested Table)。

表 3.1 对 3 种集合类型进行了比较。

表 3.1　三种集合类型比较

集合类型	元素数量	索引类型	密度	未初始化状态	何处定义	作为复合类型的元素
关联数组	不指定	String or PLS_INTEGER	不定	空	PL/SQL 块或包	否
可变数组	指定	INTEGER	密集	NULL	PL/SQL 块、包或模式	模式下定义可以
嵌套表	不指定	INTEGER	开始密集，可变为稀疏	NULL	PL/SQL 块、包或模式	模式下定义可以

其中：

① 元素数量。如果指定了元素数量，则该数量为集合中的最大元素数。如果未指定元素数，则集合中的最大元素数为索引类型的数量上限。

② 密度。密集集合的元素之间没有间隙，第一个和最后一个元素之间的每个元素都必须已定义并具有一个值（该值可以为 NULL，除非该元素具有 NOT NULL 约束）。稀疏集合的元素之间可以存在间隙。

③ 未初始化状态。空集合，一个没有元素的集合。要添加元素，需要调用集合的 EXTEND 方法；NULL 集合，一个不存在的集合。要将 NULL 集合更改为存在的状态，必须通过将其设置为空或为其分配非空值来初始化它（通过调用相关构造函数）。无法通过直接调用 EXTEND 方法来初始化一个 NULL 集合。

④ 何处定义。PL/SQL 块中定义的集合类型属于本地类型。它仅支持在块中使用，并且仅当块位于独立子程序或包子程序中时才会被存储在数据库中。包规范中定义的集合类型是公共类型。可以通过包名限定（package_name.type_name）从包外部引用它。它会一直存储在数据库中，直到删除包。在模式中定义的集合类型是独立类型，可以使用 CREATE TYPE 创建它。它存储在数据库中，直到使用 DROP TYPE 删除它。

⑤ 作为复合类型的元素：要成为复合类型的元素类型，集合类型必须是独立集合类型，即定义在模式中的集合类型。

3.1.1 关联数组

关联数组是一组键值对。每个键都是一个唯一的索引，用于定位与之相关联的值，语法格式如下。

```
variable_name(index)
```

索引的数据类型可以是字符串类型（VARCHAR2、VARCHAR、STRING 或 LONG）或 PLS_INTEGER。其中数据是按索引排序顺序存储，而不是按创建顺序存储。

与数据库表相同的是：

① 关联数组在填充之前为空，但不为 NULL。

② 关联数组可以容纳不定量的元素，可以在不知道其位置的情况下访问这些元素。

与数据库表不同的是：

① 关联数组不需要磁盘空间或网络操作。

② 关联数组不能使用 DML 语句操作。

1. 关联数组的声明

示例 3.1：以字符串为索引的关联数组。

功能描述：本例定义一种按字符串索引的关联数组，并声明该类型的变量，用 3 个元素填充该关联数组，然后更改其中一个元素的值，并打印值（数据是按索引排序顺序存储，而不是按创建顺序）。

程序代码如下。

```
DECLARE
```

```
  --Associative array indexed by string:
  TYPE ass_type IS TABLE OF INT
  INDEX BY VARCHAR2(64);
  age ass_type;
  i VARCHAR2(64);

BEGIN
  --Add elements (key-value pairs) to associative array:
  age('zs') := 20;
  age('ls') := 30;
  age('ww') := 40;
  --Change value associated with key 'Smallville':
  age('zs') := 25;
  --Print associative array:
  i  := age.FIRST;
  WHILE i IS NOT NULL LOOP
      RAISE NOTICE '% is % years old', i, age(i);
      i := age.NEXT(i);
  END LOOP;
END;
```

程序运行结果如下。

```
NOTICE:ls is 30 years old
NOTICE:ww is 40 years old
NOTICE:zs is 25 years old
```

示例 3.2：函数返回以 PLS_INTEGER 为索引的关联数组。

功能描述：本例定义了一种以 PLS_INTEGER 为索引的关联数组，以及一个返回该关联数组类型的函数。

程序代码如下。

```
DECLARE
  TYPE area_of_circle IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
  area area_of_circle;    --result variable
  num INT = 3;
  FUNCTION get_area_of_circle (
      num INT
  ) RETURN area_of_circle
  IS
      s area_of_circle;
  BEGIN
      FOR i IN 1..num LOOP
          s(i) := 2 * 3.14 * i * i;
      END LOOP;
      RETURN s;
  END;
BEGIN
  area= get_area_of_circle (num);
```

```
    FOR i in 1..num LOOP
        RAISE NOTICE 'area(%) = %', i, area(i);
    END LOOP;
END;
```

程序运行结果如下。

```
NOTICE:  area(1) = 6.28
NOTICE:  area(2) = 25.12
NOTICE:  area(3) = 56.52
```

2. 关联数组的适用情况

关联数组适用于以下情况。

① 查找一个相对较小的表，每次调用子程序或初始化声明子程序的包时，都可以在内存中构造它。

② 向数据库服务器传递集合和从数据库服务器传递集合。

关联数组用于临时数据存储。要使关联数组在数据库会话的生命周期内保持存在，请在包规范中声明它，并在包体中对其进行填充赋值。

3.1.2 可变数组

可变数组是一个数组，其元素数为从零(空)到声明的最大值之间，大小不等。

访问可变数组变量的元素，可以使用语法 variable_name(index)。指数的下限为 1，上限是当前元素的数量。上限在添加或删除元素时会发生变化，但不能超过声明时指定的最大值。从数据库中存储和检索可变数组时，其索引和元素顺序保持对应。

未初始化的可变数组变量是一个空集合。必须通过构造函数或者为其赋予一个非空的值来初始化它。

1. 可变数组的声明

示例 3.3：可变数组。

功能描述：本例定义了一个本地可变数组类型，声明了该类型的变量，并使用构造函数进行初始化，定义了一个打印可变数组的存储过程。之后调用该存储过程 3 次：分别为初始化变量后，更改两个元素的值后，以及使用构造函数更改所有元素的值后。

程序代码如下。

```
DECLARE
  --VARRAY type
  TYPE VARRAY_TYPE IS VARRAY(4) OF VARCHAR2(15);
  --varray variable initialized with constructor:
  class VARRAY_TYPE := VARRAY_TYPE('zs', 'ls', 'ww', 'zl');

  PROCEDURE show_class (heading VARCHAR2) IS
  BEGIN
      RAISE NOTICE '%', heading;
```

```
      FOR i IN 1..4 LOOP
          RAISE NOTICE '%.%' ,i ,class(i);
      END LOOP;
      RAISE NOTICE '---';
  END;

BEGIN
    show_class('2001 Class:');
    --Change values of two elements
    class(3) := 'xx';
    class(4) := 'xm';
    show_class('2005 Class:');

    --Invoke constructor to assign new values to varray variable:
    class := VARRAY_TYPE('xz', 'xw', 'xc', 'xx');
    show_class('2009 Class:');
END;
```

程序运行结果如下。

```
NOTICE:2001 Class:
NOTICE:1.zs
NOTICE:2.ls
NOTICE:3.ww
NOTICE:4.zl
NOTICE:---
NOTICE:2005 Class:
NOTICE:1.zs
NOTICE:2.ls
NOTICE:3.xx
NOTICE:4.xm
NOTICE:---
NOTICE:2009 Class:
NOTICE:1.xz
NOTICE:2.xw
NOTICE:3.xc
NOTICE:4.xx
NOTICE:---
```

2. 可变数组的适用情况

可变数组适用于以下情况。

① 已知元素的最大数量。

② 需要按顺序访问元素。

因为必须同时存储或检索所有元素，所以，可变数组一般不适用于拥有大量元素的情况。

3.1.3　嵌套表

在数据库中，嵌套表是一种可以不指定顺序来存储未指定数量行的类型。

从数据库中检索嵌套表值到 PL/SQL 嵌套表变量时，PL/SQL 会从 1 开始为行提供连续索引。使用这些索引，可以访问嵌套表变量的各个行。语法是 variable_name(index)。从数据库中存储和检索嵌套表时，嵌套表的索引和行顺序可能不稳定。

当添加或删除元素时，嵌套表变量占用的内存量可以动态地增加或减少。

未初始化的嵌套表变量是一个 NULL 集合。必须通过构造函数或为其赋予非空值进行初始化。

1. 嵌套表的声明

示例 3.4：块内部的嵌套表类型。

功能描述：本例定义一个块内部的嵌套表类型，然后声明该类型的变量(使用构造函数初始化)，并定义一个打印嵌套表的存储过程。之后调用该存储过程 3 次：初始化变量后、更改一个元素的值后，以及使用构造函数更改所有元素的值后。在第 2 次构造函数调用之后，嵌套表只有 2 个元素，引用元素 3 会引发错误。

程序代码如下。

```
DECLARE
  --nested table type
  TYPE NESTTAB_TYPE IS TABLE OF VARCHAR2(15);
  --nested table variable initialized with constructor:
  fruits NESTTAB_TYPE := NESTTAB_TYPE('Apple', 'Orange', 'Banana', 'PEAR');

  PROCEDURE show_fruits (heading VARCHAR2) IS
  BEGIN
      RAISE NOTICE '%', heading;

      FOR i IN fruits.FIRST .. fruits.LAST LOOP
          RAISE NOTICE '%', fruits(i);
      END LOOP;

      RAISE NOTICE '---';
  END;

BEGIN
  show_fruits('Initial Values:');
  --Change value of one element
  fruits(3) := 'Watermelon';
  show_fruits('Current Values:');
  --Change entire table
  fruits := NESTTAB_TYPE('Strawberry', 'Pineapple');
  show_fruits('Current Values:');
END;
```

程序运行结果如下。

```
NOTICE:  Initial Values:
NOTICE:  Apple
NOTICE:  Orange
```

```
NOTICE:  Banana
NOTICE:  PEAR
NOTICE:  ---
NOTICE:  Current Values:
NOTICE:  Apple
NOTICE:  Orange
NOTICE:  Watermelon
NOTICE:  PEAR
NOTICE:  ---
NOTICE:  Current Values:
NOTICE:  Strawberry
NOTICE:  Pineapple
NOTICE:  ---
```

示例 3.5：独立的嵌套表类型。

功能描述：本例定义了一个独立的类型 nest_type 和 1 个独立的存储过程 show_nesttype，用来打印该类型的变量。匿名块声明 1 个类型为 nest_type 的嵌套表变量，用构造函数将其初始化为空，变量初始化后，使用构造函数更改所有元素的值，两次调用 show_nesttype。

程序代码如下。

```
CREATE OR REPLACE TYPE nest_type IS TABLE OF NUMBER;

CREATE OR REPLACE PROCEDURE show_nesttype (nt nest_type) AUTHID DEFINER IS
  i  NUMBER;
BEGIN
  i := nt.FIRST;

  IF i IS NULL THEN
      RAISE NOTICE 'nest type is empty';
  ELSE
      WHILE i IS NOT NULL LOOP
          RAISE NOTICE 'nt(%) = %', i, nt(i);
          i := nt.NEXT(i);
      END LOOP;
   END IF;

  RAISE NOTICE '---';
END show_nesttype;

DECLARE
    nt nest_type := nest_type();    --nested table variable initialized to empty
BEGIN
    show_nesttype(nt);
    nt := nest_type(1,3,99,1001);
    show_nesttype(nt);
END;
```

程序运行结果如下。

```
NOTICE:  nest type is empty
NOTICE:  ---
NOTICE:  nt(1) = 1
NOTICE:  nt(2) = 3
NOTICE:  nt(3) = 99
NOTICE:  nt(4) = 1001
NOTICE:  ---
```

2. 嵌套表的适用情况

嵌套表适用于以下情况。

① 未指定元素的数量。

② 索引值是不连续的。

③ 需要删除或更新某些元素,但不能同时删除或更新所有元素。

3.1.4 集合的构造函数

集合的构造函数(constructor)是一个系统定义的函数,与集合类型同名,返回值为对应集合类型。

构造函数调用的语法格式如下。

```
collection_type ( [ value [, value ]... ] )
```

如果参数列表为空,则构造函数返回一个空集合。否则,构造函数将返回包含指定值的集合。

可以在变量声明和块的执行部分中将返回的集合分配给相同类型的集合变量。

示例 3.6:将可变数组变量初始化为空。

功能描述:本例调用了一个构造函数 2 次,将可变数组变量 class 在声明中初始化为空,并在块的执行部分为其提供新值。应用存储过程 show_class 打印变量 class 的元素值。为了确定集合何时为空,show_class 使用了集合的 COUNT 方法。

程序代码如下。

```
DECLARE
  TYPE VARRAY_TYPE IS VARRAY(4) OF VARCHAR2(15);
  class VARRAY_TYPE := VARRAY_TYPE();    --initialize to empty

  PROCEDURE show_class (heading VARCHAR2)
  IS
  BEGIN
      RAISE NOTICE '%', heading;

      IF class.COUNT = 0 THEN
          RAISE NOTICE 'Empty';
      ELSE
```

```
            FOR i IN 1..4 LOOP
              RAISE NOTICE '%.%' ,i ,class(i);
            END LOOP;
        END IF;

      RAISE NOTICE '---';
    END;
BEGIN
    show_class('Class:');
    class := VARRAY_TYPE('xx', 'xm', 'xz', 'xh');
    show_class('Class:');
END;
```

程序运行结果如下。

```
NOTICE:  Class:
NOTICE:  Empty
NOTICE:  ---
NOTICE:  Class:
NOTICE:  1.xx
NOTICE:  2.xm
NOTICE:  3.xz
NOTICE:  4.xh
NOTICE:  ---
```

3.1.5　集合变量赋值

可以通过以下方式为集合变量赋值。

① 调用构造函数来创建集合，并将其分配给集合变量。

② 使用赋值语句将另一个现有集合变量的值赋值给它。

③ 将其作为 OUT 或 IN OUT 参数传递给子程序，然后在子程序内赋值。

要为集合变量的标量元素赋值，使用 collection_variable_name(index)语法引用这些元素，并为其赋值。

1. 数据类型兼容

只有当集合变量具有相同的数据类型时，才能将集合分配给集合变量。只有元素类型相同，则无法互相赋值。

示例 3.7：数据类型兼容的集合赋值。

功能描述：在本例中，可变数组类型 VARRAY_TYPE1 和 VARRAY_TYPE2 有相同的元素类型 TEXT。集合变量 varray1 和 varray2 有相同的数据类型 VARRAY_TYPE1，但是集合变量 varray3 是数据类型 VARRAY_TYPE2。varray1 给 varray2 赋值成功，但 varray1 给 varray3 赋值失败。

程序代码如下。

```
DECLARE
  TYPE VARRAY_TYPE1 IS VARRAY(5) OF TEXT;
```

```
  TYPE VARRAY_TYPE2 IS VARRAY(5) OF TEXT;

  varray1 VARRAY_TYPE1 := VARRAY_TYPE1('Jones', 'Wong', 'Marceau');
  varray2 VARRAY_TYPE1;
  varray3 VARRAY_TYPE2;
BEGIN
  varray2 := varray1;  --ok
  varray3 := varray1;  --error
END;
```

程序运行结果如下。

```
ERROR:  can not cast from Collection type to Collection type.
```

2. 给可变数组和嵌套表变量赋 NULL 值

可以给可变数组或嵌套表变量赋 NULL 值或相同数据类型的 NULL 集合。任一赋值都会使变量为空。

示例 3.8：给嵌套表变量赋 NULL 值。

功能描述：在本例中，先将嵌套表变量 city_names 初始化为非空值；然后为其分配空集合置空；最后将其重新初始化为不同的非空值。

程序代码如下。

```
DECLARE
  TYPE cnames_tab IS TABLE OF VARCHAR2(30);
  --Initialized to non-null value
  city_names cnames_tab := cnames_tab(
    'Beijing','Shanghai','Guangzhou','Shenzhen');
  --Not initialized, therefore null
  empty_set cnames_tab;

  PROCEDURE show_city_names_status IS
  BEGIN
      IF city_names IS NULL THEN
          RAISE NOTICE 'city_names is null.';
      ELSE
          RAISE NOTICE 'city_names is not null.';
      END IF;
  END  show_city_names_status;

BEGIN
  show_city_names_status;
  city_names := empty_set;  --Assign null collection to city_names.
  show_city_names_status;
  city_names := cnames_tab (
    'Shipping','Sales','Finance','Payroll');  --Re-initialize city_names
  show_city_names_status;
END;
```

程序运行结果如下。

```
NOTICE:  city_names is not null.
NOTICE:  city_names is null.
NOTICE:  city_names is not null.
```

3.1.6　多维集合

集合只有一个维度，但可以通过使集合的元素为另一个集合来构造一个多维集合。

示例 3.9：二维可变数组。

功能描述：在本例中，nva 是一个二维可变数组（元素为整型可变数组的数组）。

程序代码如下。

```
DECLARE
  TYPE v1 IS VARRAY(10) OF INTEGER;    --varray of integer
  nva v1 := v1(2,3,5);
  TYPE v2 IS VARRAY(10) OF v1;         --varray of varray of integer
  nva v2 := v2(va, v1(1,2,3), v1(4,5), va);
  i INTEGER;
  va1 v1;
BEGIN
  i := nva(2)(3);
  RAISE NOTICE 'i = %', i;
  nva.EXTEND;
  nva(5) := v1(6, 7);                  --replace inner varray elements
  nva(4) := v1(8,9,10,10000);          --replace an inner integer element
  nva(4)(4) := 1;                      --replace 43345 with 1
  nva(4).EXTEND;                       --add element to 4th varray element
  nva(4)(5) := 98;                     --store integer 89 there
END;
```

程序运行结果如下。

```
NOTICE:  i = 3
```

示例 3.10：嵌套表类型的嵌套表及整型可变数组的嵌套表。

功能描述：在本例中，ntb1 是元素为字符串嵌套表的嵌套表，ntb2 是元素为整型可变数组的嵌套表。

程序代码如下。

```
DECLARE
  TYPE tab1 IS TABLE OF VARCHAR2(20); --nested table of strings
  vtb1 tab1 := tab1('t1', 't2');
  TYPE ntab1 IS TABLE OF tab1;         --nested table of nested tables of strings
  vntb1 ntab1 := ntab1(vtb1);
  TYPE tv1 IS VARRAY(10) OF INTEGER;   --varray of integers
  TYPE ntb2 IS TABLE OF tv1;           --nested table of varrays of integers
```

```
  vntb2 ntb2 := ntb2(tv1(1,2), tv1(3,4,5));
BEGIN
  vntb1.EXTEND;
  vntb1(2) := vntb1(1);
  vntb1.DELETE(1);                    --delete first element of vntb1
  vntb1(2).DELETE(1);    --delete first string from second table in nested table
END;
```

3.1.7 集合的比较

要确定一个集合变量与另一个集合变量的大小，必须定义在该内容中小于的含义，并编写一个返回 TRUE 或 FALSE 的函数。

嵌套表除了相等和不相等外，无法使用其他关系运算符对两个集合变量进行比较。这种限制也适用于隐式比较。例如，集合变量不能出现在 DISTINCT、GROUP BY 或 ORDER BY 子句中。

1. 将可变数组和嵌套表变量与 NULL 进行比较

可以使用 IS［NOT］NULL 运算符，将可变数组和嵌套表变量与 NULL 进行比较，但是不能使用关系运算符等于(＝)和不等于(＜＞，!＝或^＝)进行比较。

示例 3.11：将可变数组和嵌套表变量与 NULL 进行比较。

程序代码如下。

```
DECLARE
  TYPE VARRAY_TYPE IS VARRAY(4) OF VARCHAR2(15);       --VARRAY type
  class VARRAY_TYPE;                                   --varray variable
  TYPE NEST_TYPE IS TABLE OF TEXT;                     --nested table type
  fruits NEST_TYPE := NEST_TYPE('Pear', 'Apple');      --nested table variable

BEGIN
  IF class IS NULL THEN
    RAISE NOTICE 'class IS NULL';
  ELSE
    RAISE NOTICE 'class IS NOT NULL';
  END IF;

  IF fruits IS NOT NULL THEN
    RAISE NOTICE 'fruits IS NOT NULL';
  ELSE
    RAISE NOTICE 'fruits IS NULL';
  END IF;
END;
```

程序结果运行如下。

```
NOTICE:  class IS NULL
NOTICE:  fruits IS NOT NULL
```

2. 嵌套表的相等和不相等比较

两个嵌套表变量只有在拥有相同元素集的情况下才相等(任何顺序都可以)。

如果两个嵌套表变量具有相同的嵌套表类型,并且该嵌套表类型没有记录类型的元素,则可以使用关系运算符比较这两个变量的相等(=)或不相等(<>,!=,^=)。

示例 3.12:嵌套表的相等和不相等比较。

程序代码如下。

```
DECLARE
  TYPE NEST_TYPE IS TABLE OF VARCHAR2(30); -- element type is not record type
  fruits1 NEST_TYPE :=
    NEST_TYPE('Apple','Banana','Pear','Orange');
  fruits2 NEST_TYPE :=
    NEST_TYPE('Apple','Pear','Banana','Orange');
  fruits3 NEST_TYPE :=
    NEST_TYPE('Apple','Pineapple','Strawberry');
BEGIN
  IF fruits1 = fruits2 THEN
    RAISE NOTICE 'dept_names1 = dept_names2';
  END IF;
  IF fruits2 != fruits3 THEN
    RAISE NOTICE 'dept_names2 != dept_names3';
  END IF;
END;
```

程序运行结果如下。

```
NOTICE:  dept_names1 = dept_names2
NOTICE:  dept_names2 != dept_names3
```

3.1.8　集合方法

集合方法是 PL/SQL 子程序,既可以是一个返回集合信息的函数,也可以是一个对集合进行操作的过程。集合方法使集合更易于使用和维护。

表 3.2 描述了各种集合方法的情况。

表 3.2　各种集合方法

方　法	类　型	描　述
DELETE	存储过程	从集合中删除元素
TRIM	存储过程	从可变数组或嵌套表的末尾删除元素
EXTEND	存储过程	从可变数组或嵌套表的末尾增加元素
FIRST	函数	返回集合的第一个索引
LAST	函数	返回集合的最后一个索引
COUNT	函数	返回集合元素的数量

续表

方　法	类　型	描　述
LIMIT	函数	返回集合可以包含的最大的元素数量
PRIOR	函数	返回指定索引之前的索引
NEXT	函数	返回指定索引的下一个索引

集合方法的基本调用语法格式如下。

```
collection_name.method
```

集合方法可以出现在任何调用 PL/SQL 子程序(函数或过程)的地方,SQL 语句除外。在子程序中,集合作为参数程序的参数继承了对应集合的属性。可以将集合方法应用于此类参数。对于 varray 数组参数,无论参数模式如何,LIMIT 的值总是源于参数类型定义。

示例 3.13:不同集合类型方法的调用。

程序代码如下。

```
DECLARE
  TYPE ASS_TYPE IS TABLE of int index by VARCHAR2(15);     --关联数组
  TYPE VARRAY_TYPE IS VARRAY(5) OF int;                    --可变数组
  TYPE NESTTAB_TYPE IS TABLE OF int;                       --嵌套表
  v_indextab ASS_TYPE;                                     --声明关联数组类型的变量
  v_array VARRAY_TYPE := VARRAY_TYPE(10,20);               --声明可变数组变量
  v_nesttab NESTTAB_TYPE := NESTTAB_TYPE(10,20,30);        --声明嵌套表类型变量

  --输出关联数组
  PROCEDURE SHOW_ASSTYPE(at ASS_TYPE) IS
    i TEXT;
  BEGIN
    i := at.FIRST;
    IF i IS NULL THEN
      DBMS_OUTPUT.PUT_LINE('asstype is empty');
    ELSE
      WHILE i IS NOT NULL LOOP
        RAISE NOTICE 'at.(%) = %', i, at(i);
        i := at.NEXT(i);
      END LOOP;
    END IF;

      RAISE NOTICE '---';
    END;

  --输出可变数组
  PROCEDURE SHOW_ARRAY(ay VARRAY_TYPE)
    IS
    BEGIN
      IF ay IS NULL THEN
```

```
        RAISE NOTICE 'Does not exist';
      ELSIF ay.FIRST IS NULL THEN
        RAISE NOTICE 'Has no members';
      ELSE
        FOR i IN ay.FIRST..ay.LAST LOOP
          RAISE NOTICE '%.%', i, ay(i);
        END LOOP;
      END IF;

      RAISE NOTICE '---';
    END;

  --输出嵌套表
  PROCEDURE SHOW_NESTTYPE (nt NESTTAB_TYPE) IS
    i NUMBER;
    BEGIN
    i:= nt.FIRST;
      IF i IS NULL THEN
        RAISE NOTICE 'nest type is empty';
      ELSE
        WHILE i IS NOT NULL LOOP
          RAISE NOTICE 'nt(%) = %', i, nt(i);
          i := nt.NEXT(i);
        END LOOP;
       END IF;
      RAISE NOTICE '---';
  END;
BEGIN
  RAISE NOTICE '==============关联数组==============';
  v_indextab('A') := 10;
  v_indextab('B') := 20;
  v_indextab('C') := 30;
  v_indextab('D') := 40;
  v_indextab('E') := 50;
  SHOW_ASSTYPE(v_indextab);
  v_indextab.DELETE('C');
  SHOW_ASSTYPE(v_indextab);
  v_indextab.DELETE('A','D');
  SHOW_ASSTYPE(v_indextab);
  v_indextab.DELETE;
  SHOW_ASSTYPE(v_indextab);

  RAISE NOTICE '==============可变数组==============';
  v_array.EXTEND(3,1);
  SHOW_ARRAY(v_array);
  v_array.TRIM(2);
  SHOW_ARRAY(v_array);

  RAISE NOTICE '==============嵌套表==============';
  v_nesttab.EXTEND(2,1);
```

```
    SHOW_NESTTYPE(v_nesttab);
    v_nesttab.DELETE(5);
    SHOW_NESTTYPE(v_nesttab);
    v_nesttab.EXTEND;
    SHOW_NESTTYPE(v_nesttab);
    v_nesttab.TRIM;
    SHOW_NESTTYPE(v_nesttab);
END;
```

程序运行结果如下。

```
NOTICE:==============关联数组==============
NOTICE:at.(A) = 10
NOTICE:at.(B) = 20
NOTICE:at.(C) = 30
NOTICE:at.(D) = 40
NOTICE:at.(E) = 50
NOTICE:---
NOTICE:at.(A) = 10
NOTICE:at.(B) = 20
NOTICE:at.(D) = 40
NOTICE:at.(E) = 50
NOTICE:---
NOTICE:at.(E) = 50
NOTICE:---
NOTICE:---
NOTICE:==============可变数组==============
NOTICE:1.10
NOTICE:2.20
NOTICE:3.10
NOTICE:4.10
NOTICE:5.10
NOTICE:---
NOTICE:1.10
NOTICE:2.20
NOTICE:3.10
NOTICE:---
NOTICE:==============嵌套表==============
NOTICE:nt(1) = 10
NOTICE:nt(2) = 20
NOTICE:nt(3) = 30
NOTICE:nt(4) = 10
NOTICE:nt(5) = 10
NOTICE:---
NOTICE:nt(1) = 10
NOTICE:nt(2) = 20
NOTICE:nt(3) = 30
NOTICE:nt(4) = 10
NOTICE:---
```

```
NOTICE:nt(1) = 10
NOTICE:nt(2) = 20
NOTICE:nt(3) = 30
NOTICE:nt(4) = 10
NOTICE:nt(6) = <NULL>
NOTICE:---
NOTICE:nt(1) = 10
NOTICE:nt(2) = 20
NOTICE:nt(3) = 30
NOTICE:nt(4) = 10
NOTICE:---
```

3.2　记录类型

3.2.1　记录类型概述

在记录类型中，内部元素可以有不同的数据类型，每一个元素称为字段。可以通过字段名称访问记录变量的每个字段，语法是 variable_name.field_name。可以把记录类型想象成面向对象中的类。

3.2.2　声明记录类型

可以通过以下 3 种方式创建记录类型变量。

(1) 使用 RECORD 语句自定义记录类型，然后使用该记录类型声明记录类型变量。

在 PL/SQL 块中定义的 RECORD 类型是本地类型，仅在块中有效。

在包规范中定义的 RECORD 类型是公共项。可以通过包名限定的方式(package_name.type_name)在包外引用。它会一直存储在数据库中，直到通过 DROP PACKAGE 语句删除包。

不能在模式中创建 RECORD 类型，因此，RECORD 类型不可以作为 ADT 属性数据类型。

为了定义记录类型，需要指定名称和定义字段，字段默认值为 NULL。可以给字段加 NOT NULL 约束，这种情况下必须指定一个非 NULL 的初始值。如果没有 NOT NULL 约束，则这个非 NULL 初始值可选。

(2) 通过%ROWTYPE 声明一个记录类型，该变量可以表示数据库表或视图的完整行或部分行。

%ROWTYPE 属性允许声明一个 record 变量，该变量表示数据库表或视图的完整行或部分行。要声明始终代表数据库表或视图的整行 record 变量，语法格式如下。

```
variable_name table_or_view_name%ROWTYPE;
```

对于表或视图的每一列，record 变量都有一个相同名称和数据类型的字段。

(3) 通过%TYPE 属性，可以声明与之前声明的变量或列具有相同数据类型的数据项(而不需要知道该类型具体是什么)。

3.2.3 使用记录类型

声明记录类型后,可以用来定义记录类型变量。以下各示例演示各记录类型如何使用。

示例 3.14：记录类型定义和变量声明。

程序代码如下。

```
DECLARE
  TYPE StuRecTyp IS RECORD (
    stu_id  NUMBER(4) NOT NULL := 1,
    stu_name  VARCHAR2(30) NOT NULL := 'xm',
    score   NUMBER(5,2) = 88.88
  );

stu_rec StuRecTyp;
BEGIN
  RAISE NOTICE 'stu_id = %', stu_rec.stu_id;
  RAISE NOTICE 'stu_name = %', stu_rec.stu_name;
  RAISE NOTICE 'score = %', stu_rec.score;
END;
```

程序运行结果如下。

```
NOTICE:  stu_id = 1
NOTICE:  stu_name = xm
NOTICE:  score = 88.88
```

示例 3.15：声明 RECORD(指定默认值)。

程序代码如下。

```
DECLARE
  TYPE Rec IS RECORD (a NUMBER, b NUMBER);
  r Rec := (0,1);
BEGIN
  raise notice 'r = %',r;
  raise notice 'r.a = %',r.a;
  raise notice 'r.b = %',r.b;
END;
```

程序运行结果如下。

```
NOTICE:  r = (0,1)
NOTICE:  r.a = 0
NOTICE:  r.b = 1
```

示例 3.16：嵌套的记录类型。

功能描述：本例定义了 2 个记录类型：stu_info_rec 和 stu_rec。类型 stu_rec 有 1 个字段为 stu_info_rec 类型。

程序代码如下。

```
DECLARE
  TYPE stu_info_rec IS RECORD (
    id NUMBER(4),
    name VARCHAR2(30)
  );

  TYPE stu_rec IS RECORD (
    stu_info  stu_info_rec,          --nested record
    score NUMBER(5,2)
  );

  student1 stu_rec;
BEGIN
  student1.stu_info.id := 1;
  student1.stu_info.name := 'xx';
  student1.score := 88;

  RAISE NOTICE '%, %, % ', student1.stu_info.id, student1.stu_info.name,
student1.score;
END;
```

程序运行结果如下。

```
NOTICE:  1, xx, 88
```

示例 3.17：包含 Varray 数组的记录类型。

功能描述：本例定义了 Varray 数组类型 stu_info_rec 和记录类型 stu_rec。类型 stu_rec 有 1 个字段为 stu_info_rec 类型。

程序代码如下。

```
DECLARE
  TYPE stu_info_rec IS VARRAY(2) OF VARCHAR2(20);
  TYPE stu_rec IS RECORD (
    stu_info  stu_info_rec := stu_info_rec('xx',18),
    score NUMBER(5,2)
  );

  student1 stu_rec;
BEGIN
  student1.score := 88;

  RAISE NOTICE '%, %, %', student1.stu_info(1), student1.stu_info(2),student1.
score;
END;
```

程序运行结果如下。

```
NOTICE:  1, xx, 88
```

示例 3.18：%ROWTYPE 变量表示完整的数据库表行。

功能描述：本例声明了 1 个 record 变量，该变量表示表 customers 的 1 行，为其字段赋值，并打印。

程序代码如下。

```
DECLARE
  cust_rec customers%ROWTYPE;
BEGIN
  --Assign values to fields:
  cust_rec.custid   := 1;
  cust_rec.custname := 'zs';
  cust_rec.gender := '男';

  --Print fields:
  RAISE NOTICE 'cust_rec.custid = %, cust_rec.custname = %, cust_rec.gender = %
', cust_rec.custid, cust_rec.custname, cust_rec.gender;
END;
```

程序运行结果如下。

```
NOTICE:cust_rec.custid = 1, cust_rec.custname = zs, cust_rec.gender = 男
```