Lab 9. Records

Databases II

PL/SQL supports three kinds of record types: table-based, cursor-based, and user-defined.

9.1 Record Types

A record structure is somewhat similar to a row of a database table. Each data item is stored in a field with its own name and datatype. For example, suppose you have various data about a company, such as its name, address, and number of employees. A record containing a field for each of these items allows you to treat the company as a logical unit, making it easier to organize and represent the company's information.

TABLE-BASED AND CURSOR-BASED RECORDS

The %ROWTYPE attribute enables you to create table-based and cursor-based records. It is similar to the %TYPE attribute that is used to define scalar variables.
An example of a table-based record:

EXAMPLE

```
 DECLARE
   course_rec course%ROWTYPE;
BEGIN
  SELECT *
    INTO course_rec
    FROM course
   WHERE course_no = 25;
  DBMS_OUTPUT.PUT_LINE ('Course No: '||course_rec.course_no);
  DBMS_OUTPUT.PUT_LINE ('Course Description: '||course_rec.description);
  DBMS_OUTPUT.PUT_LINE ('Prerequisite: '|| course_rec.prerequisite);
END;
```

The `course_rec` record has the same structure as a row from the COURSE table. As a result, there is no need to reference individual record fields when the SELECT INTO statement populates the `course_rec` record.
An example of a cursor-based record:

EXAMPLE

```
DECLARE
CURSOR student_cur IS
 SELECT first_name, last_name, registration_date
   FROM student
  WHERE rownum <= 4;
student_rec student_cur%ROWTYPE;
BEGIN
  OPEN student_cur;
  LOOP
  FETCH student_cur INTO student_rec;
  EXIT WHEN student_cur%NOTFOUND;
  DBMS_OUTPUT.PUT_LINE ('Name: '|| student_rec.first_name||' '|| student_rec.last_name );
  DBMS_OUTPUT.PUT_LINE ('Registration Date: '|| student_rec.registration_date);
  END LOOP;
END;
```

Because a cursor-based record is defined based on the rows returned by a cursor's select statement, its declaration must be preceded by a cursor declaration. In other words, *a cursor-based record is dependent on a particular cursor and cannot be declared before its cursor.*

USER-DEFINED RECORDS

If you may need to create a record that is not based on any table or any one cursor, PL/SQL provides a user-defined record type that gives you complete control over the record structure.

The general syntax for creating a user-defined record is :

```
 TYPE type_name IS RECORD
    (field_name1 datatype1 [NOT NULL] [ := DEFAULT EXPRESSION],
     field_name2 datatype2 [NOT NULL] [ := DEFAULT EXPRESSION],
     ...
     field_nameN datatypeN [NOT NULL] [ := DEFAULT EXPRESSION]);
record_name TYPE_NAME;
```

First, a record structure is defined using the TYPE statement, where `type_name` is the name of the record type that is used in the second step to declare the actual record. Enclosed in parentheses are declarations of each record field, with its name and datatype. You may also specify a NOT NULL constraint and/or assign a default value. Second, the actual record is declared based on the type specified in the preceding step.

EXAMPLE

```
DECLARE
  TYPE time_rec_type IS RECORD
     (curr_date DATE,
      curr_day  VARCHAR2(12),
      curr_time VARCHAR2(8) := '00:00:00');
  time_rec TIME_REC_TYPE;
BEGIN
  SELECT sysdate
    INTO time_rec.curr_date
    FROM dual;
  time_rec.curr_day  := TO_CHAR(time_rec.curr_date, 'DAY');
  time_rec.curr_time := TO_CHAR(time_rec.curr_date, 'HH24:MI:SS');
  DBMS_OUTPUT.PUT_LINE ('Date: '||time_rec.curr_date);
  DBMS_OUTPUT.PUT_LINE ('Day:  '||time_rec.curr_day);
  DBMS_OUTPUT.PUT_LINE ('Time: '||time_rec.curr_time);

END;
```

When declaring a record type, you may specify a NOT NULL constraint for individual fields. It is important to note that such fields must be initialized.

EXAMPLE

```
DECLARE
  TYPE sample_type IS RECORD
     (field1 NUMBER(3),
      -- initialize a NOT NULL field
      field2 VARCHAR2(3) NOT NULL := 'ABC');
  sample_rec sample_type;
BEGIN
  sample_rec.field1 := 10;
  DBMS_OUTPUT.PUT_LINE ('sample_rec.field1 = '||sample_rec.field1);
  DBMS_OUTPUT.PUT_LINE ('sample_rec.field2 = '||sample_rec.field2);
END;
```

RECORD COMPATIBILITY

You have seen that a record is defined by its name, structure, and type. However, it is important to realize that two records may have the same structure yet be of a different type. As a result, certain restrictions apply to the operations between different record types.

FOR EXAMPLE

```
DECLARE
   TYPE name_type1 IS RECORD
      (first_name VARCHAR2(15),
       last_name  VARCHAR2(30));
   name_rec1 name_type1;
   -- name_rec2 name_type2;
   name_rec2 name_type1;
BEGIN
   name_rec1.first_name := 'John';
   name_rec1.last_name  := 'Smith';
   name_rec2 := name_rec1; -- no longer illegal assignment

END;
```

It is important to note that the assignment restriction just mentioned applies to user-defined records. In other words, *you can assign a table-based or cursor-based record to a user-defined record as long as they have the same structure.* Consider the following example:

EXAMPLE

```
 DECLARE
   CURSOR course_cur IS
      SELECT *
        FROM course
       WHERE rownum <= 4;
   -- table-based record
   -- user-defined record
   course_rec1 course%ROWTYPE;
   course_rec2 course_cur%ROWTYPE;
   -- cursor-based record
   TYPE course_type IS RECORD
      (course_no     NUMBER(38),
       description   VARCHAR2(50),
       cost          NUMBER(9,2),
       prerequisite  NUMBER(8),
       created_by    VARCHAR2(30),
       created_date  DATE,
       modified_by   VARCHAR2(30),
       modified_date DATE);
   course_rec3 course_type;
 BEGIN
   -- Populate table-based record
   SELECT *
     INTO course_rec1
     FROM course
    WHERE course_no = 10;
   -- Populate cursor-based record
   OPEN course_cur;
   LOOP
      FETCH course_cur INTO course_rec2;
      EXIT WHEN course_cur%NOTFOUND;
   END LOOP;
   course_rec1 := course_rec2;
   course_rec3 := course_rec2;
 END;
```

In this example, each record is a different type; however, they are compatible with each other because all records have the same structure. As a result, this example does not cause any syntax errors.

9.2 Nested Records

PL/SQL allows you to define nested records. These are records that contain other records and collections. The record that contains a nested record or collection is called an enclosing record.

EXAMPLE

```
DECLARE
   TYPE name_type IS RECORD
      (first_name VARCHAR2(15),
       last_name  VARCHAR2(30));
   TYPE person_type IS (
       name name_type,
       street VARCHAR2(50),
       city   VARCHAR2(25),
       state  VARCHAR2(2),
       zip    VARCHAR2(5));
   person_rec person_type;
```

This code fragment contains two user-defined record types. The second user-defined record type, `person_type`, is a nested record type because its field name is a record of the `name_type` type.

EXAMPLE

```
DECLARE
  TYPE name_type IS RECORD
     (first_name VARCHAR2(15),
      last_name  VARCHAR2(30));
  TYPE person_type IS RECORD
     (name   name_type,
      street VARCHAR2(50),
      city   VARCHAR2(25),
      state  VARCHAR2(2),
      zip    VARCHAR2(5));
  person_rec person_type;
BEGIN
  SELECT first_name, last_name, street_address, city, state, zip
    INTO person_rec.name.first_name, person_rec.name.last_name, person_rec.street,
         person_rec.city, person_rec.state, person_rec.zip
    FROM student
    JOIN zipcode USING (zip)
   WHERE rownum < 2;
  DBMS_OUTPUT.PUT_LINE ('Name: '|| person_rec.name.first_name||'
                        '||person_rec.name.last_name);
  DBMS_OUTPUT.PUT_LINE ('Street: '||person_rec.street);
  DBMS_OUTPUT.PUT_LINE ('City:   '||person_rec.city);
  DBMS_OUTPUT.PUT_LINE ('State:  '||person_rec.state);
  DBMS_OUTPUT.PUT_LINE ('Zip:    '||person_rec.zip);

END;
```

In this example, the `person_rec` record is a user-defined nested record. Therefore, to reference its field name that is a record with two fields, the following syntax is used:

```
enclosing_record.(nested_record or nested_collection).field_name
```

9.3 Collections of Records

PL/SQL also lets you define a collection of records :

EXAMPLE

```
DECLARE
  CURSOR name_cur IS
     SELECT first_name, last_name
       FROM student
      WHERE ROWNUM <= 4;
  TYPE name_type IS TABLE OF name_cur%ROWTYPE
     INDEX BY BINARY_INTEGER;
  name_tab  name_type;
  v_counter INTEGER := 0;
BEGIN
  FOR name_rec IN name_cur LOOP
    v_counter := v_counter + 1;
    name_tab(v_counter).first_name := name_rec.first_name;
    name_tab(v_counter).last_name := name_rec.last_name;
    DBMS_OUTPUT.PUT_LINE('First Name('||v_counter||'): '||
        name_tab(v_counter).first_name);
    DBMS_OUTPUT.PUT_LINE('Last Name('||v_counter||'): '||
        name_tab(v_counter).last_name);
  END LOOP;
END;
```

In the declaration portion of this example, you define the `name_cur` cursor, which returns the first and last names of the first four students. Next, you define an associative array type whose element type is based on the cursor defined previously using the %ROWTYPE attribute. Then you define an associative array variable and the counter that is used later to reference individual rows of the associative array. In the executable portion of the example, you populate the associative array and display its records on the screen. Consider the notation used in the example when referencing individual elements of the array:

```
name_tab(v_counter).first_name
```

Exercices:

1. Create an associative array with the element type of a user-defined record. This record should contain the first name, last name, and total number of courses that a particular instructor teaches. Display the records of the associative array on the screen.
2. Modify the script you just created. Instead of using an associative array, use a nested table.
3. Modify the script you just created. Instead of using a nested table, use a varray.
4. Create a user-defined record with four fields: course_no, description, cost, and prerequisite_rec. The last field, prerequisite_rec, should be a user-defined record with three fields: prereq_no, prereq_desc, and prereq_cost. For any ten courses that have a prerequisite course, populate the user-defined record with all the corresponding data, and display its information on the screen.