
ORACLE
Academy

Database Programming with PL/SQL

6-1

User-Defined Records

ORACLE
Academy

Objectives

- This lesson covers the following objectives:
  - Create and manipulate user-defined PL/SQL records

Purpose

- You already know how to declare and use PL/SQL record structures that correspond to the data fetched by a cursor, using the %ROWTYPE attribute
- What if you want to create and use a variable structure (called a record) that corresponds to an entire row in a table, or a view, or a join of several tables, rather than using just one or two columns?
- Or maybe you need a record structure which does not correspond to any object(s) in the database

In previous lessons, we saw a record declared with %ROWTYPE based on a cursor's structure. We will now examine a record based on a table's structure as we prepare to work with more complex record structures.

PL/SQL Records

- A PL/SQL record is a composite data type consisting of a group of related data items stored as fields, each with its own name and data type
- You can refer to the whole record by its name and/or to individual fields by their names
- Typical syntax for defining a record is shown below
- This code defines a record based on the structure of a row within the named table

```
record_name    table_name%ROWTYPE;
```

Structure of a PL/SQL Record

- You reference each field by dot-prefixing its field-name with the record-name : record_name.field_name
- For example, you reference the job_id field in the v_emp_record record as follows:
  - v_emp_record.job_id

Field1 (data type)	Field2 (data type)	Field3 (data type)	...

The Problem

- The EMPLOYEES table contains eleven columns: EMPLOYEE_ID, FIRST_NAME,...., MANAGER_ID, DEPARTMENT_ID
- You need to code a SELECT * INTO variable names FROM EMPLOYEES WHERE… in your PL/SQL subprogram
- How many scalar variables must you DECLARE to hold the column values?

ORACLE Academy

Answer: eleven, one for each column in the table.

The Problem

- That is a lot of coding, and some tables will have even more columns
- Plus, what do you do if a new column is added to the table?
- Or an existing column is dropped?

The Problem

```
DECLARE
 v_employee_id  employees.employee_id%TYPE;
 v_first_name  employees.first_name%TYPE;
 v_last_name employees.last_name%TYPE;
 v_email   employees.email%TYPE;
 ... FIVE MORE SCALAR VARIABLES REQUIRED TO MATCH THE TABLE
 v_manager_id  employees.manager_id%TYPE;
 v_department_id employees.department_id%TYPE;
BEGIN
 SELECT employee_id, first_name, ... EIGHT MORE HERE,
department_id
  INTO v_employee_id, v_first_name, ... AND HERE,
v_department_id
  FROM employees
  WHERE employee_id = 100;
END;
```

The Problem

- Look at the code again
- Wouldn’t it be easier to declare one variable instead of eleven?
- %ROWTYPE allows us to declare a variable as a record based on a particular table's structure
- Each field or component within the record will have its own name and data type based on the table's structure
- You can refer to the whole record by its name and to individual fields by their names

The Solution - Use a PL/SQL Record

- Use %ROWTYPE to declare a variable as a record based on the structure of the EMPLOYEES table
- Less code to write and nothing to change if columns are added or dropped

```
DECLARE
 v_emp_record  employees%ROWTYPE;
BEGIN
 SELECT * INTO v_emp_record
  FROM employees
 WHERE employee_id = 100;
 DBMS_OUTPUT.PUT_LINE('Email for ' || v_emp_record.first_name
||
  ' ' || v_emp_record.last_name || ' is ' ||
v_emp_record.email ||
  '@oracle.com.');
END;
```

A Record Based on Another Record

- You can use %ROWTYPE to declare a record based on another record:

```
DECLARE
 v_emp_record  employees%ROWTYPE;
 v_emp_copy_record  v_emp_record%ROWTYPE;
BEGIN
 SELECT * INTO v_emp_record
  FROM employees
 WHERE employee_id = 100;
 v_emp_copy_record := v_emp_record;

 v_emp_copy_record.salary := v_emp_record.salary * 1.2;

 DBMS_OUTPUT.PUT_LINE(v_emp_record.first_name ||
  ' ' || v_emp_record.last_name || ': Old Salary - ' ||
  v_emp_record.salary || ', Proposed New Salary - ' ||
  v_emp_copy_record.salary || '.');
END;
```

Defining Your Own Records

- What if you need data from a join of multiple tables?
- You can declare your own record structures containing any fields you like
- PL/SQL records:
  - Must contain one or more components/fields of any scalar or composite type
  - Are not the same as rows in a database table
  - Can be assigned initial values and can be defined as NOT NULL
  - Can be components of other records (nested records)

Records, even if declared using %ROWTYPE, are not the same as rows in a table.

Table rows are stored on disk and the data in a table row is permanent (unless modified using DML).

Records are stored in memory (like other variables) and the data stored in a record will persist only for the duration of the code block's execution.

Syntax for User-Defined Records

- Start with the TYPE keyword to define your record structure
- It must include at least one field and the fields may be defined using scalar data types such as DATE, VARCHAR2, or NUMBER, or using attributes such as %TYPE and %ROWTYPE
- After declaring the type, use the type_name to declare a variable of that type

```
TYPE type_name IS RECORD
 (field_declaration[,field_declaration]...);

identifier  type_name;
```

ORACLE Academy

PLSQL 6-1
User-Defined Records

Oracle scalar data types such as VARCHAR2, DATE, and NUMBER are pre-defined data types that are automatically declared (with global scope) in every Oracle database. Using the TYPE keyword, you are simply creating additional data types with a custom structure to be used within your block of code.

The *type_name* is the name you give to your custom data type.

The *field_declaration* contains the field name(s), data type(s), and any initialization values such as NOT NULL or DEFAULT.

The *identifier* is the name of the variable that is declared using the custom data type you created.

User-Defined Records: Example 1

- First, declare/define the type and a variable of that type
- Then use the variable and its components

User-Defined Records: Example 1

```
DECLARE
 TYPE person_dept IS RECORD
 (first_name      employees.first_name%TYPE,
 last_name        employees.last_name%TYPE,
     department_name departments.department_name%TYPE);
 v_person_dept_rec  person_dept;
BEGIN
 SELECT e.first_name, e.last_name, d.department_name
 INTO v_person_dept_rec
  FROM employees e JOIN departments d
 ON e.department_id = d.department_id
  WHERE employee_id = 200;
DBMS_OUTPUT.PUT_LINE(v_person_dept_rec.first_name ||
 ' ' || v_person_dept_rec.last_name || ' is in the ' ||
 v_person_dept_rec.department_name || ' department.');
END;
```

User-Defined Records: Example 2

- Here we have two custom data types, one nested within the other
- How many fields can be addressed in v_emp_dept_rec?

```
DECLARE
 TYPE dept_info_type IS RECORD
  (department_id       departments.department_id%TYPE,
   department_name     departments.department_name%TYPE);
 TYPE emp_dept_type IS RECORD
  (first_name    employees.first_name%TYPE,
   last_name     employees.last_name%TYPE,
   dept_info     dept_info_type);

 v_emp_dept_rec     emp_dept_type;
BEGIN
 ...
END;
```

ORACLE Academy

Answer: there are four fields that can be addressed in v_emp_dept_rec:

v_emp_dept_rec.first_name

v_emp_dept_rec.last_name

v_emp_dept_rec.dept_info.department_id

v_emp_dept_rec.dept_info.department_name

Declaring and Using Types and Records

- Types and records are composite structures that can be declared anywhere that scalar variables can be declared in anonymous blocks, procedures, functions, package specifications (global), package bodies (local), triggers, and so on
- Their scope and visibility follow the same rules as for scalar variables
- For example, you can declare a type (and a record based on the type) in an outer block and reference them within an inner block

Visibility and Scope of Types and Records

- The type and the record declared in the outer block are visible within the outer block and the inner block
- What will be displayed by each of the PUT_LINEs?

```
DECLARE -- outer block
 TYPE employee_type IS RECORD
 (first_name     employees.first_name%TYPE := 'Amy');
 v_emp_rec_outer    employee_type;
BEGIN
 DBMS_OUTPUT.PUT_LINE(v_emp_rec_outer.first_name);
 DECLARE -- inner block
  v_emp_rec_inner   employee_type;
 BEGIN
  v_emp_rec_outer.first_name := 'Clara';
  DBMS_OUTPUT.PUT_LINE(v_emp_rec_outer.first_name ||
 ' and ' || v_emp_rec_inner.first_name);
 END;
 DBMS_OUTPUT.PUT_LINE(v_emp_rec_outer.first_name);
END;
```


Run the code in APEX and see if your predictions are correct – you should get :

Amy

Clara and Amy

Clara

Terminology

- Key terms used in this lesson included:
  - PL/SQL record

ORACLE Academy

- PL/SQL record – a composite data type consisting of a group of related data items stored as fields, each with its own name and data type.

Summary

- In this lesson, you should have learned how to:
  - Create and manipulate user-defined PL/SQL records
  - Define a record structure using the %ROWTYPE attribute

ORACLE

Academy