Database Programming with PL/SQL

User-Defined Records

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 record structure that corresponds to a row in a table, or a view, or a join of several tables, rather than to a cursor? Or which does not correspond to any object(s) in the database?

A Problem Scenario

The EMPLOYEES table contains eleven columns: EMPLOYEE_ID, FIRST_NAME,...., MANAGER_ID, DEPARTMENT_ID.

You need to code a single-row SELECT * FROM EMPLOYEES in your PL/SQL subprogram. Because you are selecting only a single row, you do not need to declare and use a cursor.

How many scalar variables must you DECLARE to hold the column values?

A Problem Scenario: PL/SQL Code

It's a lot of coding, isn't it? And what if a new (twelfth) column is added to the table?

```
DECLARE
  v_employee_id   employees.employee_id%TYPE;
  v_first_name    employees.first_name%TYPE;
  ... -- seven more scalar variables here
  v_manager_id    employees.manager_id%TYPE;
  v_department_id employees.department_id%TYPE;
BEGIN
  SELECT employee_id, first_name, ..., department_id
    INTO v_employee_id, v_first_name, ..., v_department_id
    FROM employees
    WHERE employee_id = 100;
END;
```

A Problem Scenario: PL/SQL Code (cont.)

Look at the code again. Wouldn't it be easier to declare one variable instead of eleven (or maybe twelve in the future)?

Isn't it better to have just one large bag for all your school or college books, instead of a separate bag for each book ?

```
DECLARE
  v_employee_id   employees.employee_id%TYPE;
... -- nine more scalar variables here
  v_department_id employees.department_id%TYPE;
BEGIN ...
```

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.

PL/SQL Records (cont.)

Using %ROWTYPE implicitly declares a record whose fields match the corresponding columns by name and data type. You can reference individual fields by prefixing the field-name with the record-name.

```
... DBMS_OUTPUT.PUT_LINE(v_emp_record.salary);

... IF v_emp_record.department_id = 20 THEN ...
```

Using a PL/SQL Record

You can use %ROWTYPE with tables just as you can with cursors. And if a column is added to or dropped from the table, no change to the PL/SQL code is needed.

Now all of your books are in a single bag! Doesn't the code look better?

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

Using a PL/SQL Record: A Second Example

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 * 2;
  ...
END;
```

Defining Your Own Records

But what if your example procedure `SELECT`s from a join of several 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).

Creating a User-Defined PL/SQL Record

A record structure is a composite data type. It can consist of one or more items, most often Oracle defined scalar data types such as DATE, VARCHAR2, NUMBER, etc. You declare the type and then declare one or more variables of that type.

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

identifier   type_name;
```

field_declaration can be of any PL/SQL data type, including %TYPE, %ROWTYPE, and RECORD.

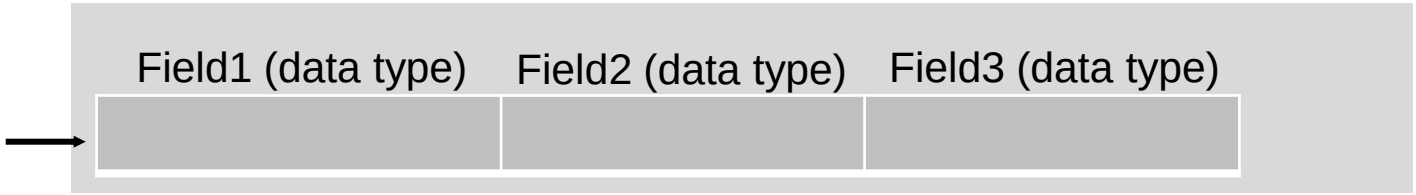
User-Defined PL/SQL Records: Example

We must declare the type as a model or template for the record, then a “real” variable of that type.

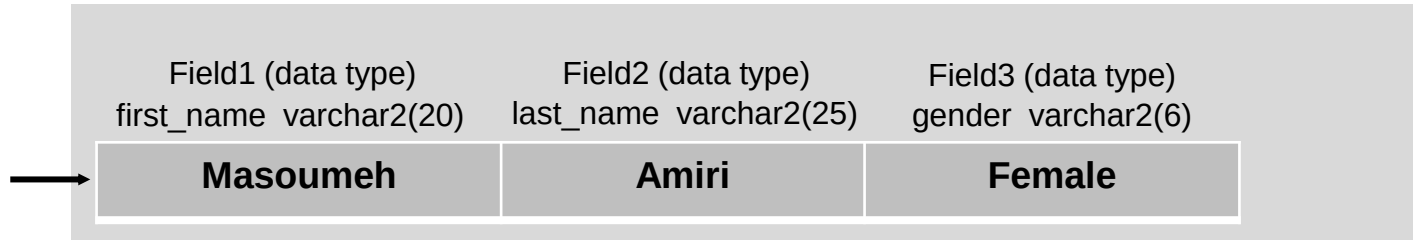
```
TYPE person_type IS RECORD
     (first_name  employees.first_name%TYPE,
      last_name   employees.last_name%TYPE,
      gender      VARCHAR2(6));
person_rec      person_type;
...
  IF person_rec.last_name ... END IF;
```

PL/SQL Record Structure

This graphic shows the structure of `PERSON_REC`.



Example (PERSON_REC):



User-Defined PL/SQL Records: Another Example

```
TYPE person_type IS RECORD
     (first_name  employees.first_name%TYPE,
      last_name   employees.last_name%TYPE,
      gender      VARCHAR2(6));
TYPE employee_type IS RECORD
     (job_id      VARCHAR2(10),
      salary      number(8,2),
      person_data person_type);

person_rec      person_type;
employee_rec    employee_type;
...
  IF person_rec.last_name ... END IF;
  employee_rec.person_data.last_name := ...;
```

User-Defined PL/SQL Records Example (cont.)

Types can contain other types (person_data is a field in employee_type).

When types contain other types, you must use multiple levels of dot-prefixing to reference individual scalar fields (employee_rec.person_data.last_name).

```
TYPE person_type IS RECORD ...;
TYPE employee_type IS RECORD (...
       person_data person_type);
person_rec      person_type;
employee_rec    employee_type;
...
  IF person_rec.last_name ... END IF;
  employee_rec.person_data.last_name := ...;
```

Where Can Types and Records Be Declared and Used?

They are composite variables and 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 follows 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 Records Example

```
DECLARE
  TYPE person_type IS RECORD
     (first_name     employees.first_name%TYPE,
      last_name      employees.last_name%TYPE,
      gender         VARCHAR2(6));
  person_rec_outer   person_type;
BEGIN
  person_rec_outer.first_name := 'Lucy';
  DECLARE
    person_rec_inner   person_type;
  BEGIN
    SELECT first_name, last_name, 'M'
    INTO person_rec_inner
    FROM employees
    WHERE employee_id = 100;
    person_rec_outer := person_rec_inner;
  END;
END;
```

This example only shows scope and does not achieve anything.

Terminology

Key terms used in this lesson included:

- PL/SQL record

Summary

In this lesson, you should have learned how to:

- Create and manipulate user-defined PL/SQL records