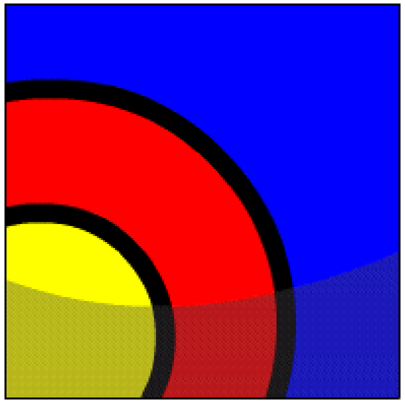
User-Defined Records

Copyright © 2008, Oracle. All rights reserved.

What Will I Learn?

In this lesson, you will learn to:

- Create and manipulate user-defined PL/SQL records



Why Learn It?

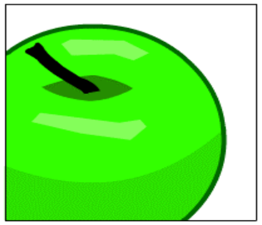
You already know how to declare and use PL/SQL record structures that correspond to the data fetched by a cursor, using the `%ROWTYPE` attribute.

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?

In this lesson you will learn how to create and use your own record structures.

Copyright © 2008, Oracle. All rights reserved.

Tell Me/Show Me



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 ?

The next slide shows the code.

Copyright © 2008, Oracle. All rights reserved.

Tell Me/Show Me

A Problem Scenario: PL/SQL Code

```
CREATE OR REPLACE PROCEDURE query_one_emp
      (p_emp_id IN employees.employee_id%TYPE)  IS
  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 = p_emp_id;
EXCEPTION
  WHEN NO_DATA_FOUND THEN ...;
END;
```

Tell Me/Show Me

How Can You Return the Results to the Calling Environment?

```
CREATE OR REPLACE PROCEDURE query_one_emp
      (p_emp_id         IN  employees.employee_id%TYPE,
       p_first_name     OUT employees.first_name%TYPE,
       ... – seven more OUT parameters here
       p_manager_id     OUT employees.manager_id%TYPE,
       p_department_id  OUT employees.department_id%TYPE)
IS
  v_employee_id   employees.employee_id%TYPE;
  v_first_name    employees.first_name%TYPE;
  ...
```

Fortunately you don’t have to do all this. Instead, you declare and use a PL/SQL record. The next slide shows how.

Copyright © 2008, Oracle. All rights reserved.

Tell Me/Show Me

Using a PL/SQL Record

```
CREATE OR REPLACE PROCEDURE query_one_emp
      (p_emp_id     IN  employees.employee_id%TYPE,
       p_emp_record OUT employees%ROWTYPE)    IS
BEGIN
  SELECT * INTO p_emp_record
    FROM employees
      WHERE employee_id = p_emp_id;
EXCEPTION
 WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('Nothing
                                               selected.');
END;
```

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 procedure is needed.

Copyright © 2008, Oracle. All rights reserved.

Tell Me/Show Me

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.

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:

```
... IF p_emp_record.salary > 25000 THEN
      RAISE_APPLICATION_ERROR
        (-20104, 'This employee earns too much!');
    END IF; ...
```

Copyright © 2008, Oracle. All rights reserved.

Tell Me/Show Me

Defining Your Own Records

But what if your example procedure SELECTs 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
- A record can be a component of another record (nested records).

Copyright © 2008, Oracle. All rights reserved.

Tell Me/Show Me

Creating a User-Defined PL/SQL Record

A record structure is a composite data type, just as `DATE`, `VARCHAR2`, `NUMBER`, and so on are Oracle-defined scalar data types. 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`.

Copyright © 2008, Oracle. All rights reserved.

ORACLE Academy

Tell Me/Show Me

User-Defined PL/SQL Records 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 := ...;
```

Copyright © 2008, Oracle. All rights reserved.

ORACLE Academy

Tell Me/Show Me

User-Defined PL/SQL Records Example (continued)

```
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 := ...;
```

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`).

Copyright © 2008, Oracle. All rights reserved.

Tell Me/Show Me

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 in a package specification. Records based on that type can be declared and used anywhere within the package, and also in the calling environment.

The next two slides show an example of this.

Copyright © 2008, Oracle. All rights reserved.

Tell Me/Show Me

```
CREATE OR REPLACE PACKAGE pers_pack IS
  TYPE person_type IS RECORD
     (first_name  employees.first_name%TYPE,
      last_name   employees.last_name%TYPE,
      gender      VARCHAR2(6));
  PROCEDURE pers_proc (p_pers_rec OUT person_type);
END pers_pack;
CREATE OR REPLACE PACKAGE BODY pers_pack IS
  PROCEDURE pers_proc (p_pers_rec OUT person_type) IS
      v_pers_rec     person_type;
    BEGIN
      SELECT first_name, last_name, 'Female' INTO
v_pers_rec
        FROM employees WHERE employee_id = 100;
    p_pers_rec := v_pers_rec;
    END pers_proc;
END pers_pack;
```

Tell Me/Show Me

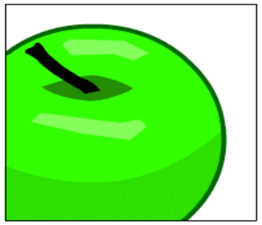
Visibility and Scope of Records Example (continued)

Now invoke the package procedure from another PL/SQL block (it could be a Java, C, or other language application):

```
DECLARE
  a_pers_rec       pers_pack.person_type;           -- 1
BEGIN
  pers_pack.pers_proc(a_pers_rec);                  -- 2
  DBMS_OUTPUT.PUT_LINE
     (a_pers_rec.first_name ||' '||a_pers_rec.gender);
END;
```

Copyright © 2008, Oracle. All rights reserved.

Tell Me/Show Me



Terminology

Key terms used in this lesson include:

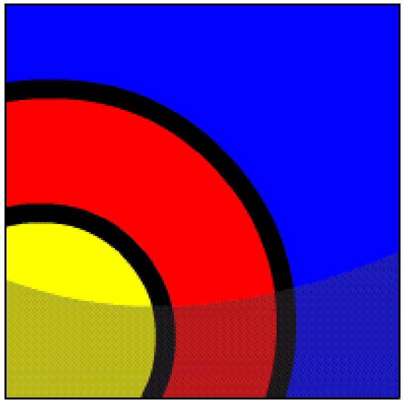
PL/SQL record

Copyright © 2008, Oracle. All rights reserved.

Summary

In this lesson, you learned to:

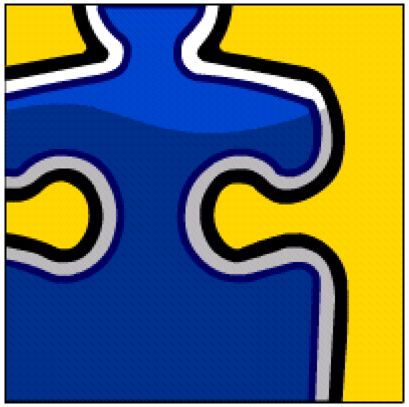
- Create and manipulate user-defined PL/SQL records



Try It/Solve It

This practice covers the following topics:

- Creating and manipulating user-defined PL/SQL records.



Copyright © 2008, Oracle. All rights reserved.