6 Working with Composite Data Types

ORACLE

Objectives

After completing this lesson, you should be able to do the following:

- Create user-defined PL/SQL records
- Create a record with the `%ROWTYPE` attribute
- Create an `INDEX BY` table
- Create an `INDEX BY` table of records
- Describe the difference between records, tables, and tables of records

Composite Data Types

- Can hold multiple values, unlike scalar types
- Are of two types:
  - PL/SQL records
  - PL/SQL collections
    - `INDEX BY` tables or associative arrays
    - Nested table
    - `VARRAY`

Composite Data Types

- Use PL/SQL records when you want to store values of different data types but only one occurrence at a time.
- Use PL/SQL collections when you want to store values of same data type.

PL/SQL Records

- Must contain one or more components of any scalar, `RECORD`, or `INDEX BY` table data type, called fields
- Are similar to structures in most 3GL languages including C and C++
- Are user defined and can be a subset of a row in a table
- Treat a collection of fields as a logical unit
- Are convenient for fetching a row of data from a table for processing

Implicit declaration with The `%ROWTYPE` Attribute

- Declare a variable according to a collection of columns in a database table or view.
- Prefix `%ROWTYPE` with the database table or view.
- Fields in the record take their names and data types from the columns of the table or view.

Syntax:

```
DECLARE
    identifier  reference%ROWTYPE;
```

EXAMPLE

```
DECLARE
     v_emp_rec employees%rowtype;
BEGIN
     select * into v_emp_rec
     from employees
     where employee_id = 137;
     /* dbms_output.put_line(v_emp_rec); geeft een foutmelding
        dbms_output.put_line(v_emp_rec.last_name); werkt foutloos */
     dbms_output.put_line('Medewerker '||v_emp_rec.first_name||' '
     ||v_emp_rec.last_name  || ' startte op '
     ||to_char(v_emp_rec.hire_date,'FMday dd month yyyy'));
END;
/
```

Advantages of using %ROWTYPE

- The number and data types of the underlying database columns need not to be known and, in fact, might change at run time
- The %ROWTYPE attribute is useful when retrieving a row with the SELECT * statement

Explicit declaration by Creating a PL/SQL Record

Syntax:

1
```
TYPE type_name IS RECORD
     (field_declaration[, field_declaration]…);
```

2
```
identifier   type_name;
```

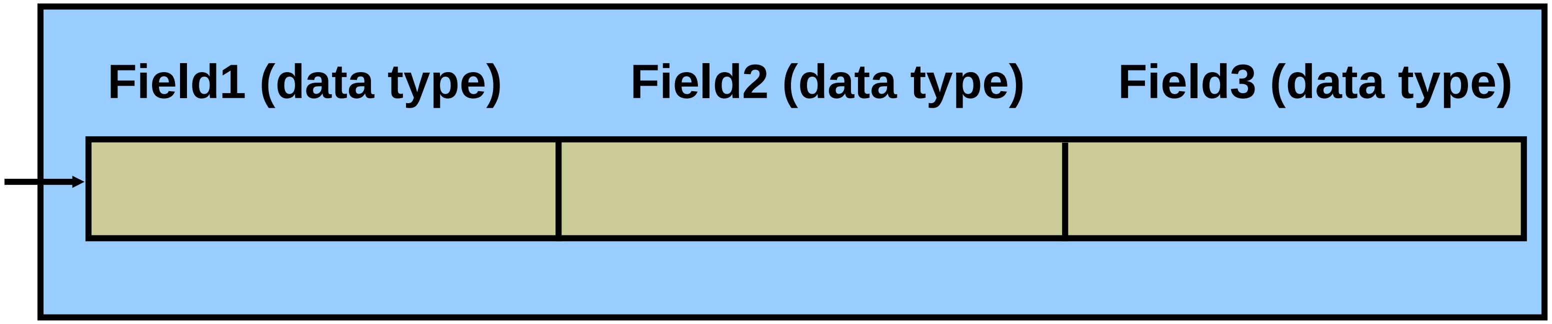
***field_declaration*:**

```
field_name {field_type | variable%TYPE
           | table.column%TYPE | table%ROWTYPE}
           [[NOT NULL] {:= | DEFAULT} expr]
```

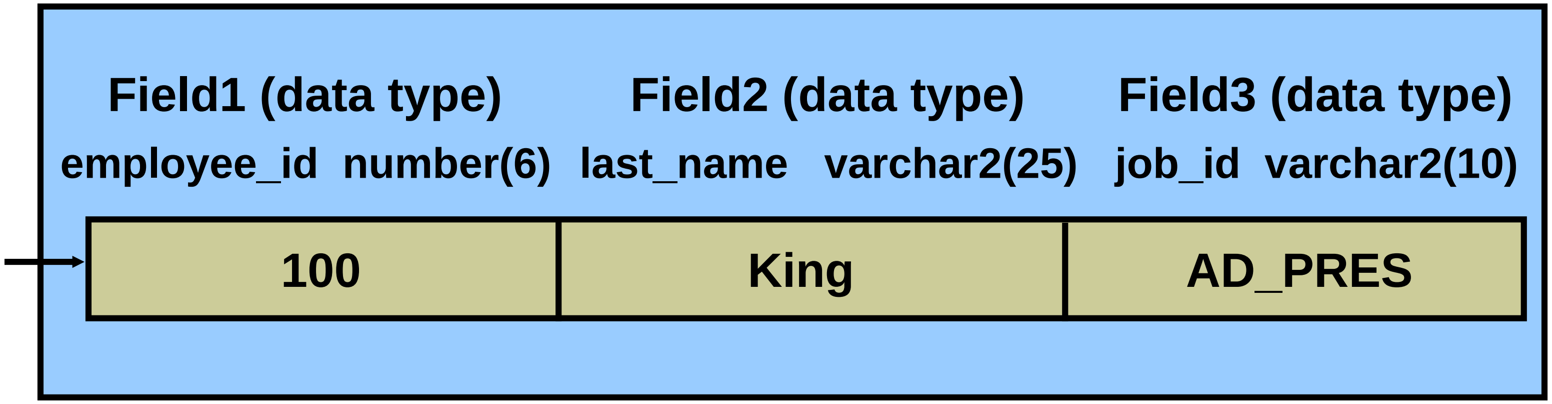
Creating a PL/SQL Record: Example

```
DECLARE
   type t_rec is record
    (v_sal number(8),
    v_minsal number(8) default 1000,
    v_hire_date employees.hire_date%type,
    v_rec1 employees%rowtype);
   v_myrec t_rec;
BEGIN
   v_myrec.v_sal := v_myrec.v_minsal + 500;
   v_myrec.v_hire_date := sysdate;
   SELECT * INTO v_myrec.v_rec1
      FROM employees WHERE employee_id = 100;
   DBMS_OUTPUT.PUT_LINE(v_myrec.v_rec1.last_name ||' '||
   to_char(v_myrec.v_hire_date) ||' '|| to_char(v_myrec.v_sal));
END;
```

PL/SQL Record Structure



Example:



The %ROWTYPE Attribute

```
...
DEFINE employee_number = 124
 DECLARE
  emp_rec   employees%ROWTYPE;
 BEGIN
  SELECT * INTO emp_rec FROM employees
  WHERE  employee_id = &employee_number;
  INSERT INTO retired_emps(empno, ename, job, mgr,
  hiredate, leavedate, sal, comm, deptno)
  VALUES (emp_rec.employee_id, emp_rec.last_name,
  emp_rec.job_id,emp_rec.manager_id,
  emp_rec.hire_date, SYSDATE, emp_rec.salary,
  emp_rec.commission_pct, emp_rec.department_id);
END;
/
```

Inserting a Record Using `%ROWTYPE`

```
...
DEFINE employee_number = 124
DECLARE
   emp_rec  retired_emps%ROWTYPE;
BEGIN
 SELECT employee_id, last_name, job_id, manager_id,
 hire_date, hire_date, salary, commission_pct,
 department_id INTO emp_rec FROM employees
 WHERE  employee_id = &employee_number;
 INSERT INTO retired_emps VALUES emp_rec;
END;
/
SELECT * FROM retired_emps;
```

Updating a Row in a Table Using a Record

```
SET SERVEROUTPUT ON
SET VERIFY OFF
DEFINE employee_number = 124
DECLARE
   emp_rec retired_emps%ROWTYPE;
BEGIN
 SELECT * INTO emp_rec FROM retired_emps;
 emp_rec.leavedate:=SYSDATE;
 UPDATE retired_emps SET ROW = emp_rec WHERE
  empno=&employee_number;
END;
/
SELECT * FROM retired_emps;
```

`INDEX BY` Tables or Associative Arrays

- Are PL/SQL structures with two columns:
  - Primary key type integer or string
  - Column of scalar or record data type
- Are unconstrained in size. However the size depends on the values the key data type can hold.

Creating an `INDEX BY` Table

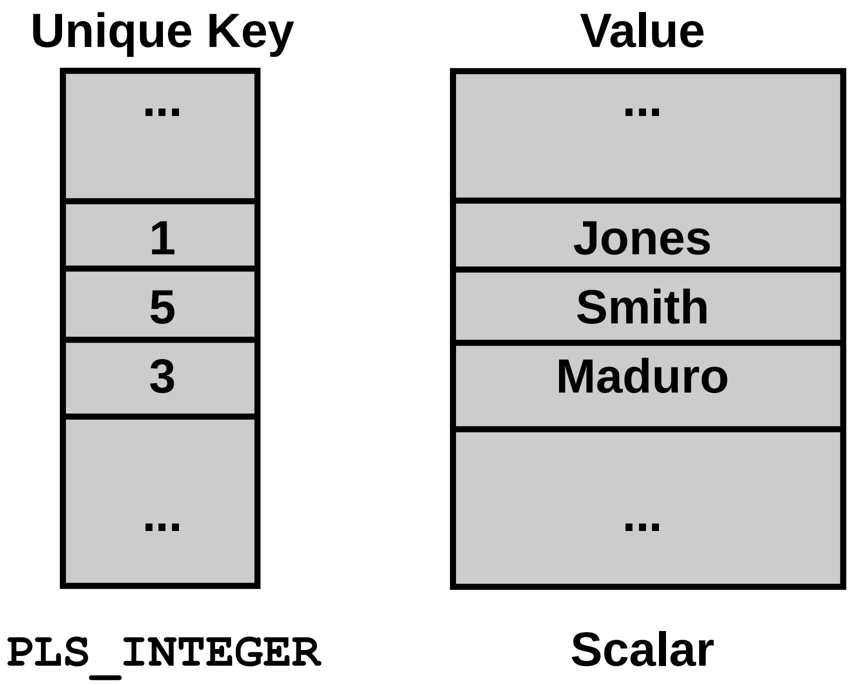
Syntax:

```
TYPE type_name IS TABLE OF
     {column_type | variable%TYPE
     | table.column%TYPE} [NOT NULL]
     | table%ROWTYPE
     [INDEX BY PLS_INTEGER | BINARY_INTEGER
      | VARCHAR2(<size>)];
identifier   type_name;
```

Declare an `INDEX BY` table to store the last names of employees.

```
...
TYPE ename_table_type IS TABLE OF
 employees.last_name%TYPE
 INDEX BY PLS_INTEGER;
 ...
ename_table ename_table_type;
```

`INDEX BY` Table Structure



Creating an `INDEX BY` Table

```
DECLARE
  TYPE ename_table_type IS TABLE OF
    employees.last_name%TYPE
    INDEX BY PLS_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
    INDEX BY PLS_INTEGER;
  ename_table         ename_table_type;
  hiredate_table      hiredate_table_type;
BEGIN
  ename_table(1)      := 'CAMERON';
  hiredate_table(8) := SYSDATE + 7;
    IF ename_table.EXISTS(1) THEN
    INSERT INTO ...
    ...
END;
/
```

Using `INDEX BY` Table Methods

The following methods make `INDEX BY` tables easier to use:

- `EXISTS`
- `COUNT`
- `FIRST` and `LAST`
- `PRIOR`
- `NEXT`
- `DELETE`

`INDEX BY` Table of Records

Define an `INDEX BY` table variable to hold an entire row from a table.

Example:

```
DECLARE
  TYPE dept_table_type IS TABLE OF
       departments%ROWTYPE
       INDEX BY PLS_INTEGER;
  dept_table dept_table_type;
  -- Each element of dept_table is a record
```

Example of `INDEX BY` Table of Records

```
SET SERVEROUTPUT ON
DECLARE
   TYPE emp_table_type IS TABLE OF
      employees%ROWTYPE INDEX BY PLS_INTEGER;
   my_emp_table  emp_table_type;
   max_count          NUMBER(3):= 104;
BEGIN
  FOR i IN 100..max_count
  LOOP
   SELECT * INTO my_emp_table(i) FROM employees
   WHERE employee_id = i;
  END LOOP;
  FOR i IN my_emp_table.FIRST..my_emp_table.LAST
  LOOP
     DBMS_OUTPUT.PUT_LINE(my_emp_table(i).last_name);
  END LOOP;
END;
/
```

Het vullen van de collection kan ook op volgende manier

```
SET SERVEROUTPUT ON
DECLARE
   TYPE emp_table_type IS TABLE OF
      employees%ROWTYPE INDEX BY PLS_INTEGER;
   my_emp_table  emp_table_type;
   max_count          NUMBER(3):= 104;
BEGIN
   SELECT * BULK COLLECT INTO my_emp_table
   FROM employees;
FOR i IN my_emp_table.FIRST..my_emp_table.LAST
  LOOP
     DBMS_OUTPUT.PUT_LINE(my_emp_table(i).last_name);
  END LOOP;
END;
/
```

Voordelen werken met collections

- Werken in geheugen is sneller dan in databank
- Bij gebruik van %ROWTYPE werkt je programma nog altijd na aanpassing van structuur in databank
- Met BULK COLLECT kan je select in 1 keer in collection geplaatst worden (betere performance)
- Vooral nuttig bij mutaties in de databank – zie volgende voorbeeld

Mutaties in de databank via collection

```
DECLARE
   TYPE emp_table_type IS TABLE OF
      employees%ROWTYPE INDEX BY PLS_INTEGER;
   my_emp_table  emp_table_type;
   v_index     pls_integer;
BEGIN
   SELECT * BULK COLLECT INTO my_emp_table
   FROM employees;
-- wijziging in collection
   v_index := my_emp_table.first;
   WHILE v_index is not null  LOOP
        IF my_emp_table(v_index).salary < 5000 AND
               my_emp_table(v_index).job_id =‘IT_PROG’
        THEN
                my_emp_table(v_index).salary:= 5000;
        END IF;
        v_index := my_emp_table.next(v_index);
   END LOOP;
Vervolg op volgende dia!
```

Mutaties in de databank via collection (vervolg)

```
-- wijziging in databank (synchroniseren van de database)
   v_index := my_emp_table.first;
   WHILE v_index is not null  LOOP
        UPDATE employees
        SET salary = my_emp_table(v_index).salary
        WHERE employee_id = my_emp_table(v_index).employee_id ;
         v_index := my_emp_table.next(v_index);
   END LOOP;
        ......
END;
/
```

Deze synchronisatie kan nog efficiënter door gebruik te maken van BULK DML maar daar gaan we niet dieper op in!

Summary

In this lesson, you should have learned how to:

- Define and reference PL/SQL variables of composite data types:
  - PL/SQL records
  - `INDEX BY` tables
  - `INDEX BY` table of records
- Define a PL/SQL record by using the `%ROWTYPE` attribute

Practice 6: Overview

This practice covers the following topics:

- Declaring `INDEX BY` tables
- Processing data by using `INDEX BY` tables
- Declaring a PL/SQL record
- Processing data by using a PL/SQL record