6

Working with Composite Data Types

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 differences among 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 the same data type.

PL/SQL Records

- Must contain one or more components (called *fields*) of any scalar, `RECORD`, or `INDEX BY` table data type
- 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

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]
```

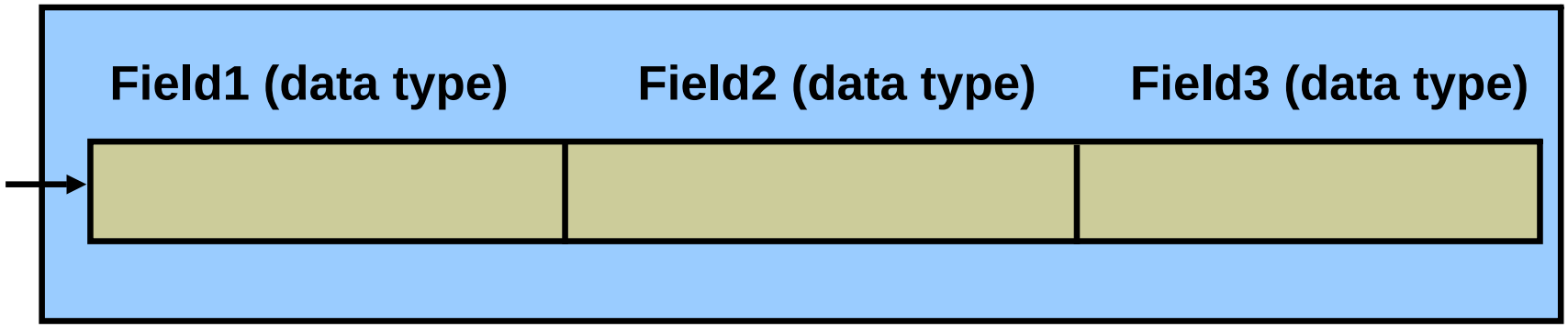
Creating a PL/SQL Record

Declare variables to store the name, job, and salary of a new employee.

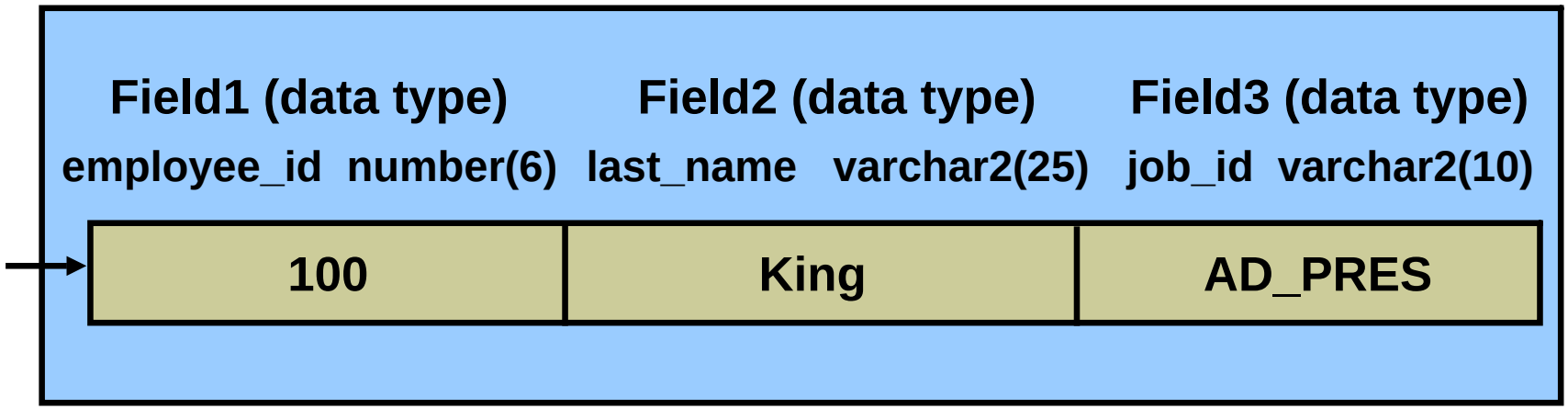
Example:

```
...
  TYPE emp_record_type IS RECORD
    (last_name    VARCHAR2(25),
     job_id       VARCHAR2(10),
     salary       NUMBER(8,2));
  emp_record      emp_record_type;
...
```

PL/SQL Record Structure



Example:



%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;
```

Advantages of Using %ROWTYPE

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

%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 by 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 by 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 of integer or string data type
  - Column of scalar or record data type
- Are unconstrained in size. However, the size depends on the values that the key data type can hold.

Creating an INDEX BY Table

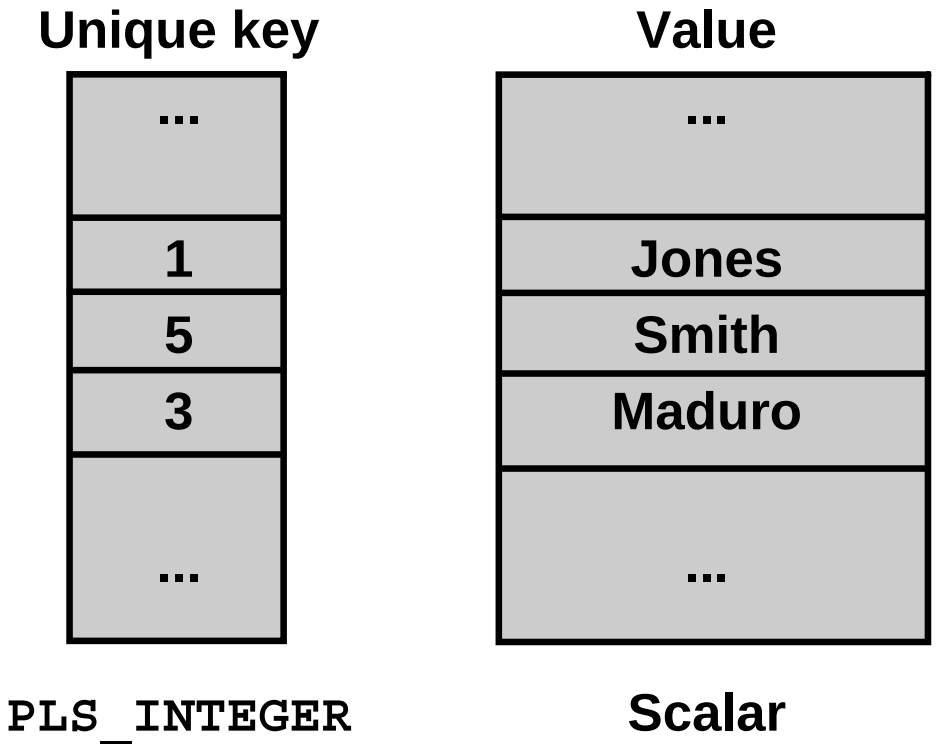
Syntax:

```
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.

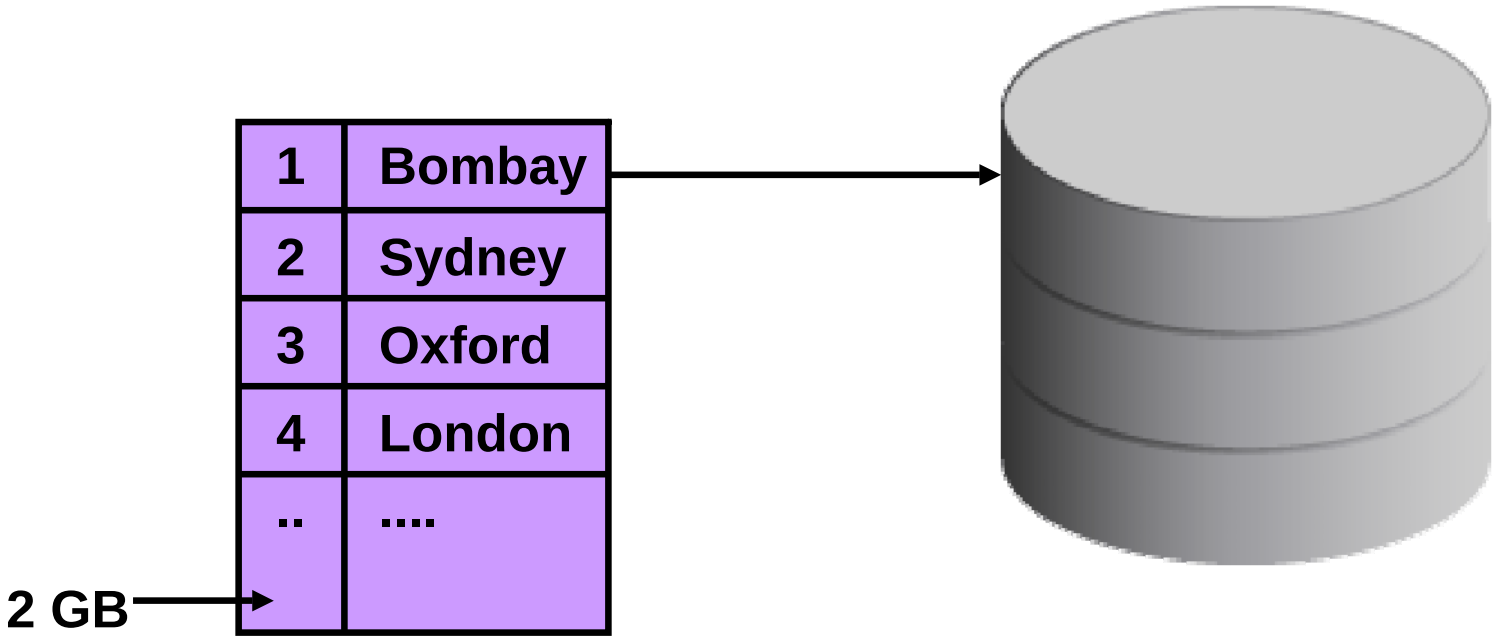
Example:

```
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
```

INDEX BY Table of Records: Example

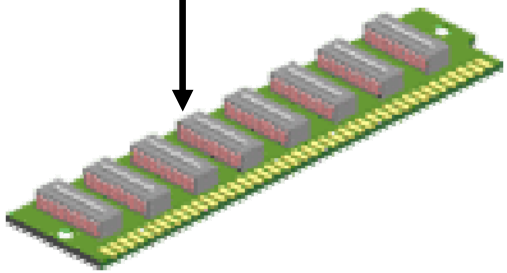
```
SET SERVEROUTPUT ON
DECLARE
   TYPE emp_table_type IS TABLE OF
      employees%ROWTYPE INDEX BY PLS_INTEGER;
   my_emp_table  emp_table_type;
   max_count          NUMBER(3):= 104;
BEGIN
  FOR i IN 100..max_count
  LOOP
   SELECT * INTO my_emp_table(i) FROM employees
   WHERE employee_id = i;
  END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

Nested Tables



VARRAY

1	Bombay
2	Sydney
3	Oxford
4	London
..	....
10	Tokyo



Summary

In this lesson, you should have learned how to:

- Define and reference PL/SQL variables of composite data types
  - PL/SQL record
  - `INDEX BY` table
  - `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