7

Working With Composite Datatypes

Objectives

After completing this lesson, you should be able to:

- Describe PL/SQL collections and records
- Create user-defined PL/SQL records
- Create a PL/SQL record with the `%ROWTYPE` attribute
- Create associative arrays
  - `INDEX BY` table
  - `INDEX BY` table of records

Course Roadmap

PLSQL

- Lesson 6: Writing Control Statements
- **Lesson 7: Working with Composite DataTypes** — You are here!
- Lesson 8: Using Explicit Cursors
- Lesson 9: Exception Handling
- Lesson 10: Stored Procedures and Functions

Agenda

- Introducing composite data types
- Using PL/SQL records
  - Manipulating data with PL/SQL records
  - Advantages of the %ROWTYPE attribute
- Using PL/SQL collections
  - Examining associative arrays
  - Introducing nested tables
  - Introducing VARRAY

Composite Data Types

- Can hold multiple values (unlike scalar types)
- Are of two types:
  - PL/SQL records
  - PL/SQL collections
    - Associative array (`INDEX BY` table)
    - Nested table
    - `VARRAY`

PL/SQL Records or Collections?

- 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 Record:

TRUE	23-DEC-98	ATLANTA	

PL/SQL Collection:

1	SMITH
2	JONES
3	BENNETT
4	KRAMER

PLS_INTEGER

VARCHAR2

Agenda

- Examining composite data types
- Using PL/SQL records
  - Manipulating data with PL/SQL records
  - Advantages of the `%ROWTYPE` attribute
- Using PL/SQL collections
  - Examining associative arrays
  - Introducing nested tables
  - Introducing `VARRAY`

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 third-generation 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

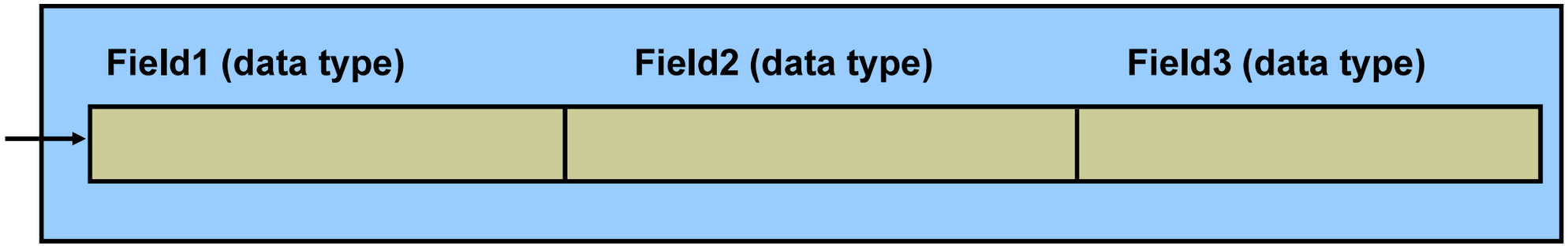
```
identifier    type_name;
```

field_declaration:

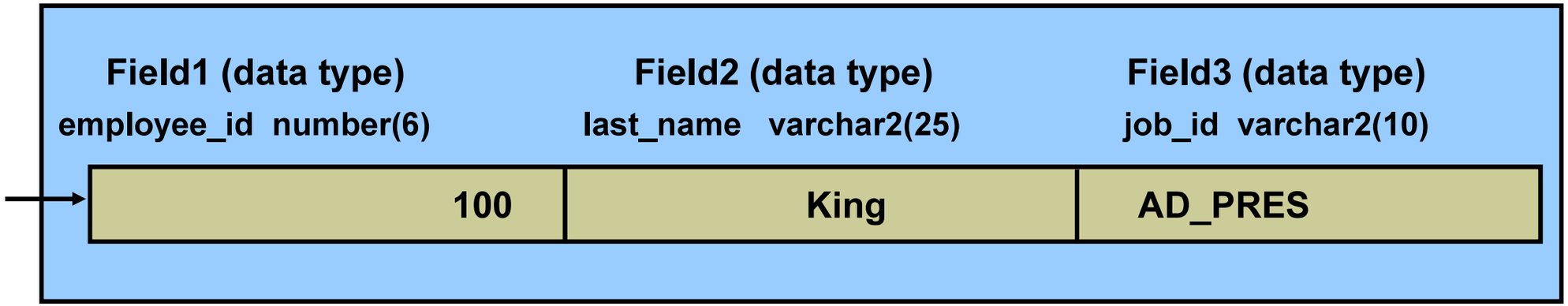
```
field_name {field_type | variable%TYPE
           | table.column%TYPE | table%ROWTYPE}
           [[NOT NULL] {:= | DEFAULT} expr]
```

PL/SQL Record Structure

Field declarations:



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.

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

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;
```

```
anonymous block completed
King 16-OCT-12 1500
```

Advantages of Using the %ROWTYPE Attribute

- 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 you want to retrieve a row with:
  - The SELECT * statement
  - Row-level INSERT and UPDATE statements

Another %ROWTYPE Attribute Example

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

```
SELECT * FROM retired_emps;
/
```

Query Result

SQL | All Rows Fetched: 1 in 0.005 seconds

	EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124	Mourgos	ST_MAN	100	16-NOV-07	16-OCT-12	5800	(null)	50

Inserting a Record by Using %ROWTYPE

```
...
DECLARE
  v_employee_number number:= 124;
  v_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 v_emp_rec FROM employees
 WHERE  employee_id = v_employee_number;
 INSERT INTO retired_emps VALUES v_emp_rec;
END;
/
SELECT * FROM retired_emps;
```

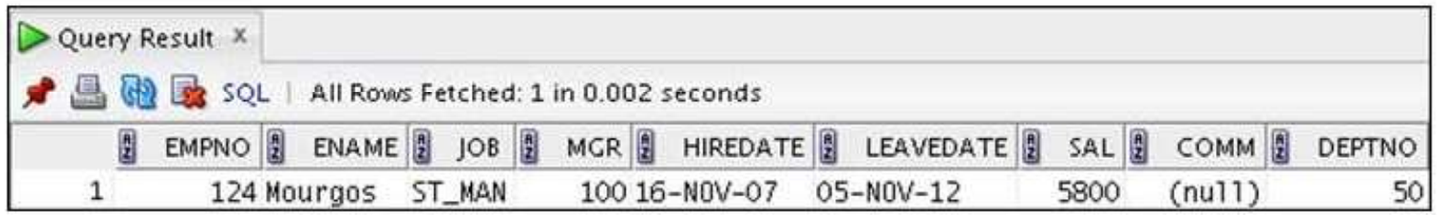
Query Result

SQL | All Rows Fetched: 1 in 0.002 seconds

	EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124	Mourgos	ST_MAN	100	16-NOV-07	16-NOV-07	5800	(null)	50

Updating a Row in a Table

```
DECLARE
  v_employee_number number:= 124;
  v_emp_rec  retired_emps%ROWTYPE;
BEGIN
  SELECT * INTO v_emp_rec FROM retired_emps WHERE
  empno = v_employee_number;
  v_emp_rec.leavedate:= CURRENT_DATE;
  UPDATE retired_emps SET ROW = v_emp_rec WHERE
  empno=v_employee_number;
END;
/
SELECT * FROM retired_emps;
```


Query Result

SQL | All Rows Fetched: 1 in 0.002 seconds

	EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124	Mourgos	ST_MAN	100	16-NOV-07	05-NOV-12	5800	(null)	50

Agenda

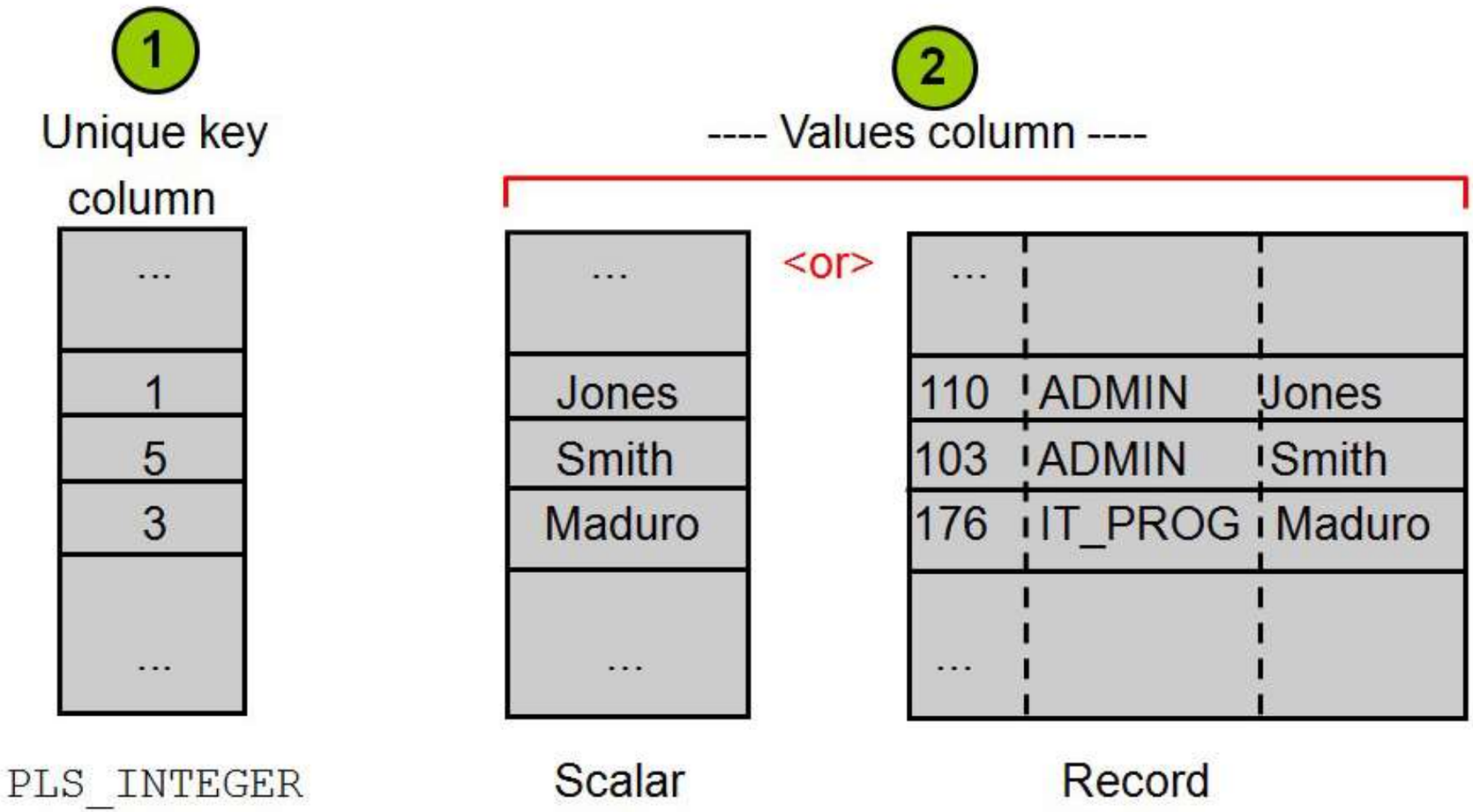
- Examining composite data types
- Using PL/SQL records
  - Manipulating data with PL/SQL records
  - Advantages of the `%ROWTYPE` attribute
- Using PL/SQL collections
  - Examining associative arrays
  - Introducing nested tables
  - Introducing `VARRAY`

Associative Arrays (INDEX BY Tables)

- An associative array is a PL/SQL collection with two columns:
  - Primary key of integer or string data type
  - Column of scalar or record data type

Key	Values
1	JONES
2	HARDEY
3	MADURO
4	KRAMER

Associative Array Structure



Steps to Create an Associative Array

- Syntax:

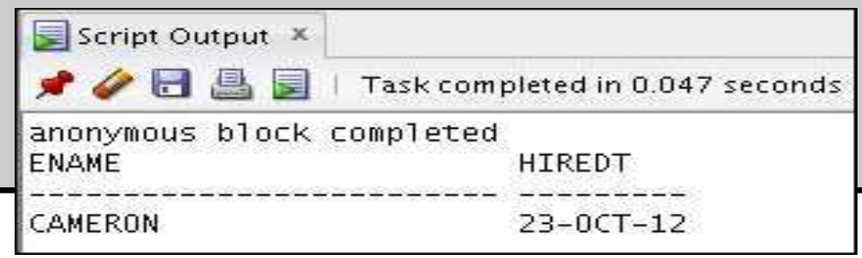
1
```
TYPE type_name IS TABLE OF
{ column_type [NOT NULL] | variable%TYPE [NOT NULL]
| table.column%TYPE [NOT NULL]
| table%ROWTYPE }
INDEX BY { PLS_INTEGER | BINARY_INTEGER
| VARCHAR2(<size>) } ;
```
2
```
identifier type_name;
```

- Example:

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

Creating and Accessing Associative Arrays

```
...
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;
/
...
```



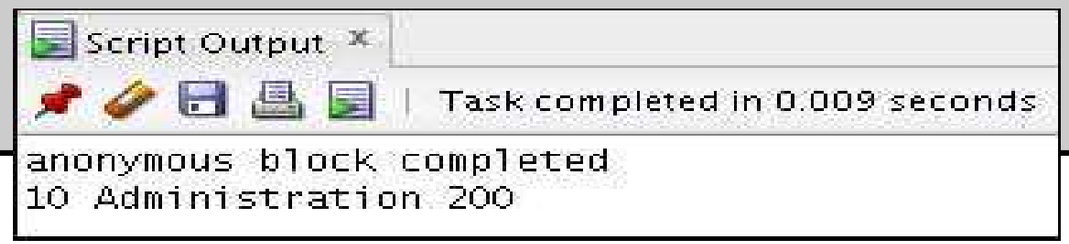
Using INDEX BY Table Methods

- The following methods make associative arrays easier to use:
  - EXISTS
  - COUNT
  - FIRST
  - LAST
  - PRIOR
  - NEXT
  - DELETE

INDEX BY Table of Records Option

- Define an associative array to hold an entire row from a table.

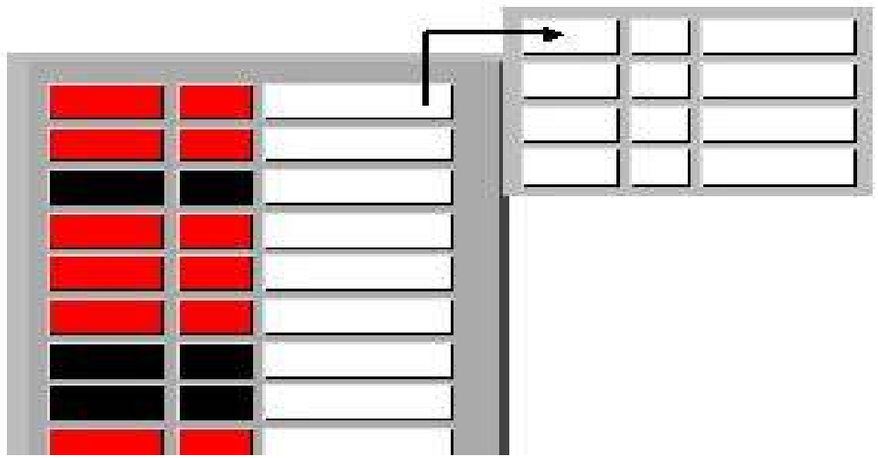
```
DECLARE
TYPE dept_table_type
IS
  TABLE OF departments%ROWTYPE INDEX BY VARCHAR2(20);
  dept_table dept_table_type;
  -- Each element of dept_table is a record
BEGIN
   SELECT * INTO dept_table(1) FROM departments
   WHERE department_id = 10;
   DBMS_OUTPUT.PUT_LINE(dept_table(1).department_id ||' '||
   dept_table(1).department_name ||' '||
   dept_table(1).manager_id);
END;
/
```



INDEX BY Table of Records Option: Example 2

```
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

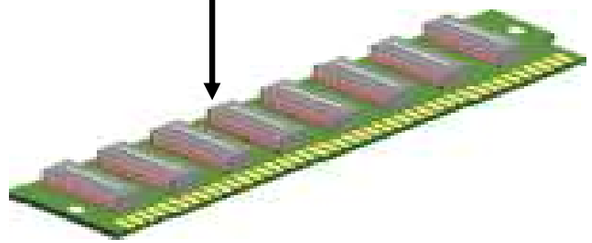


1	Bombay
2	Sydney
3	Oxford
4	London
..	....

2 GB max.

VARRAY

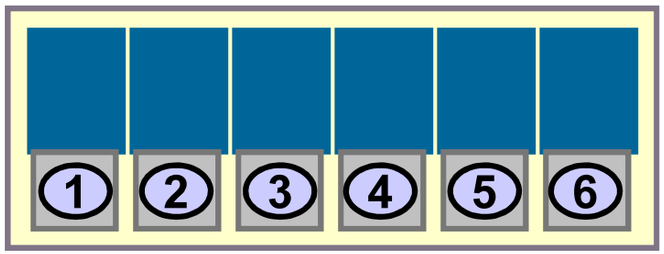
1	Bombay
2	Sydney
3	Oxford
4	London
..	....
10	Tokyo

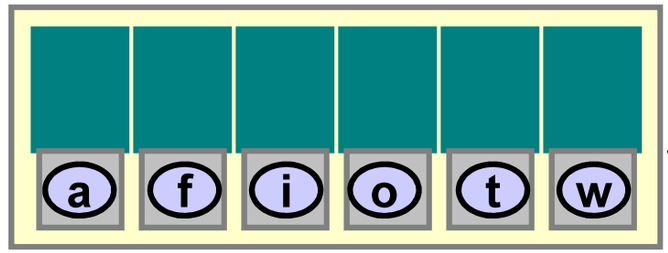


Summary of Collection Types

Associative array

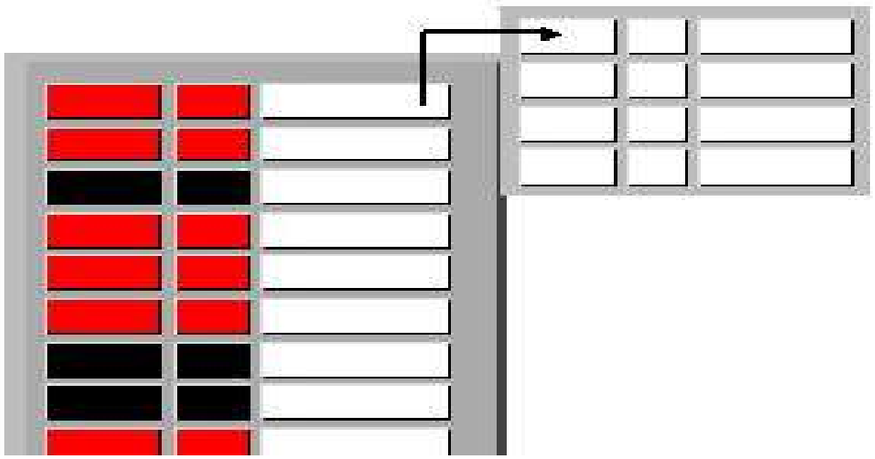
Index by
PLS_INTEGER



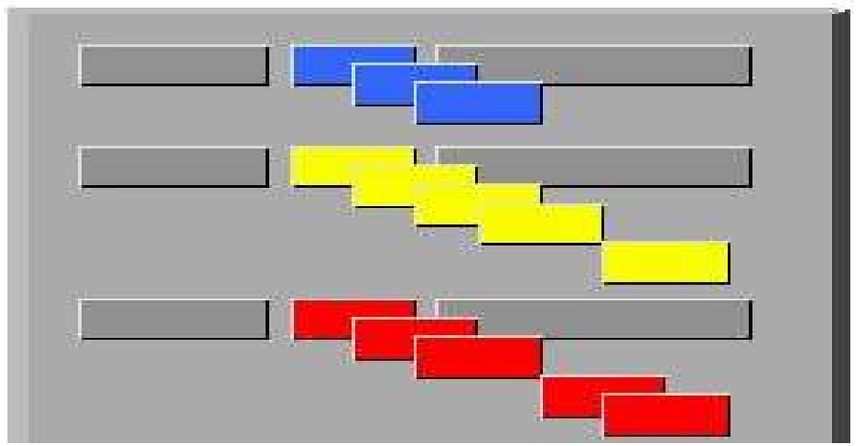


Index by
VARCHAR2

Nested table



Varray



Identify situations in which you can use the `%ROWTYPE` attribute.

a. When you are not sure about the structure of the underlying database table
b. When you want to retrieve an entire row from a table
c. When you want to declare a variable according to another previously declared variable or database column

Summary

In this lesson, you should have learned that:

- Declaring associative arrays
- Processing data by using associative arrays
- Declaring a PL/SQL record
- Processing data by using a PL/SQL record