Working with Composite Data Types

Objectives

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

- 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

Composite Data Types

Alice is the database designer of HR schema.

Alice wonders!

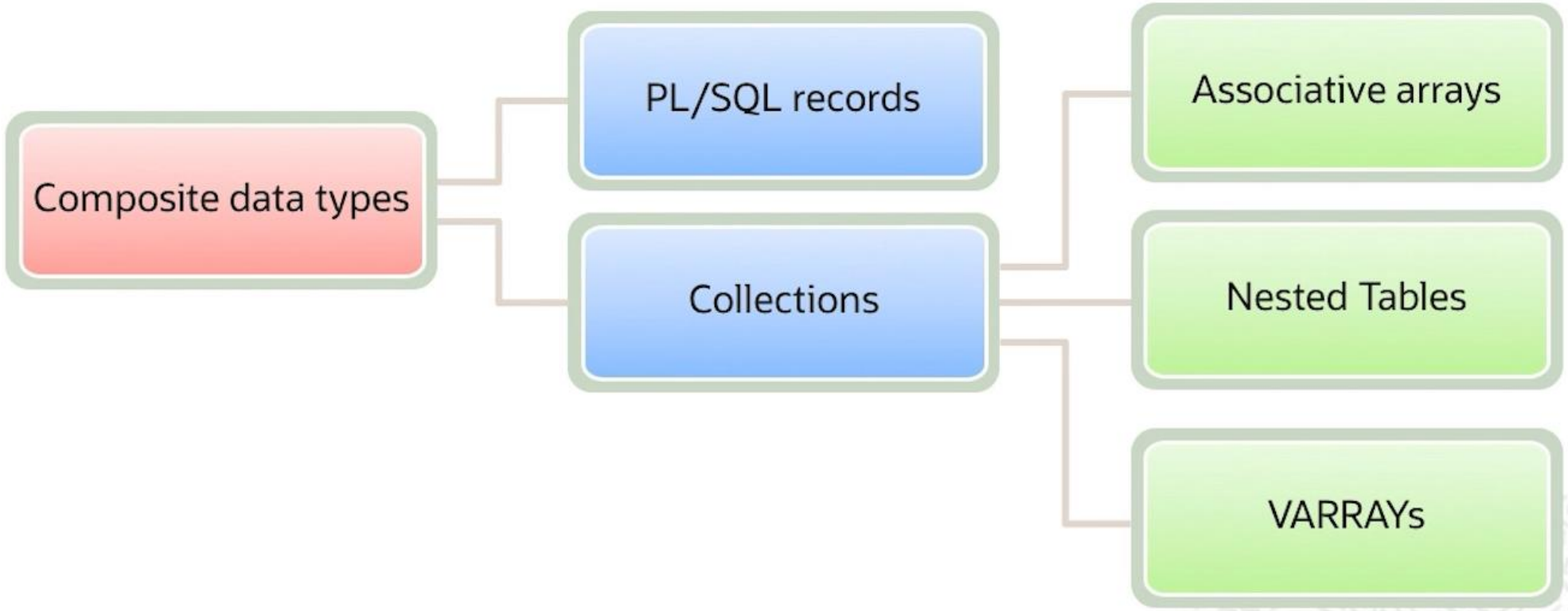
New Employees' Details

- Name ✓
- Date of Birth ✓
- Contact ✗
- Address ✗
- Designation ✓
- Department ✓
- Salary ✓

Is there a way to add more than one contact number in the same table?

Is there a way to group an employee's address as a single column?

Composite Data Types



PL/SQL Records Versus Collections

PL/SQL Records	Collections
These are used to store related but dissimilar data as a logical unit.	These are used to store data as a single unit.
Use when you want to store values of different data types that are logically related.	Use when you want to store values of the same data type.
Each element can be accessed as: `record_name.field_name`.	Each element can be accessed by its unique subscript.
Example: A record to hold employee details that are related because they provide information about a particular employee	Example: A collection to hold the emails of all employees. It may store *n* email IDs; however, email 1 is not related to email 2, and so on.

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

VARCHAR2

PLS_INTEGER

PL/SQL Records

- Update the job_id of each employee.
- Add data to the job_history table to reflect promotion.
- Modify the salary of the employee.

Creating a PL/SQL Record

Syntax:

1

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

2

```
identifiertype_name;
```

field_declaration:

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

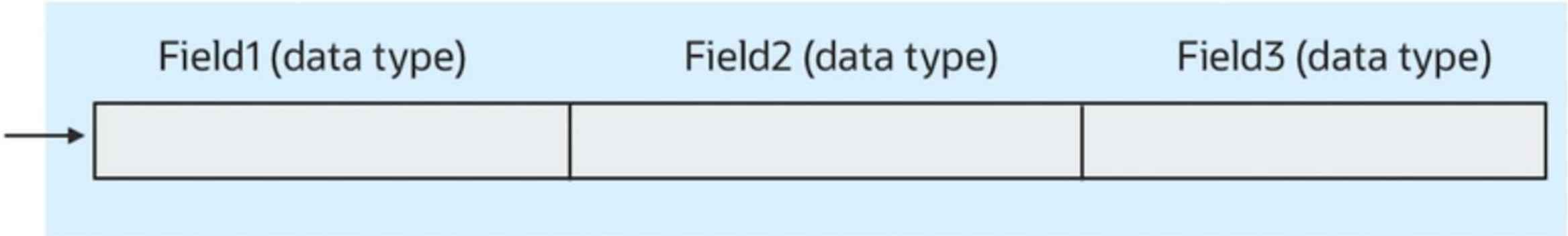
Creating a PL/SQL Record: Example

```
DECLARE
  TYPE t_rec IS RECORD
    (
     v_first_name employees.first_name%type,
     v_sal number(8),
     v_hire_date employees.hire_date%type,
     );
  v_myrec t_rec;
BEGIN
SELECT first_name,salary, hire_date INTO v_myrec
      FROM employees WHERE employee_id = 100;
  DBMS_OUTPUT.PUT_LINE('First Name:
 '||v_myrec.v_first_name ||'Salary:
 '||v_myrec.v_sal ||'Hire Date:
 '|| v_myrec.v_hire_date);
END;
```

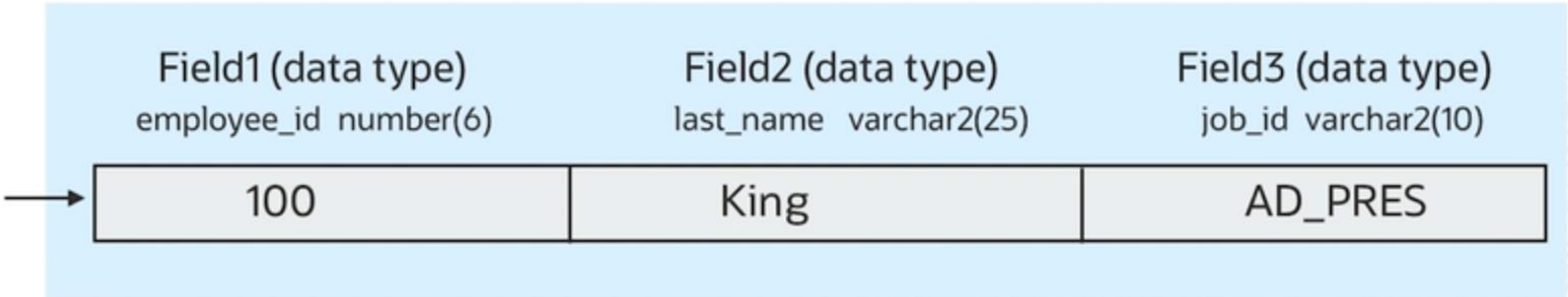
```
PL/SQL procedure successfully completed.

First Name:Steven
Salary: 24000
Hire Date: 17-JUN-11
```

PL/SQL Record Structure

Field declarations:



Example:



%ROWTYPE Attribute

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

Syntax:

Prefix %ROWTYPE with the database table or view.

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

Example:

```
DECLARE
   employees    employees%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 ||' '||
  v_myrec.v_hire_date ||' '|| v_myrec.v_sal);
END;
```

```
PL/SQL procedure successfully completed.

King  11-JUL-16  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
  - The row-level `INSERT` and `UPDATE` statements

```
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;
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 1 in 0.017 seconds

	EMP_NO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124	Mourgos	ST_MAN	100	16-NOV-15	11-JUL-16	5800	(null)	50

Updating a Row in a Table by Using a Record

```
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;
```

Script Output × Query Result ×

SQL | All Rows Fetched: 1 in 0.017 seconds

	EMP_NO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124	Mourgos	ST_MAN	100	16-NOV-15	11-JUL-16	5800	(null)	50

Agenda

- Understanding 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`

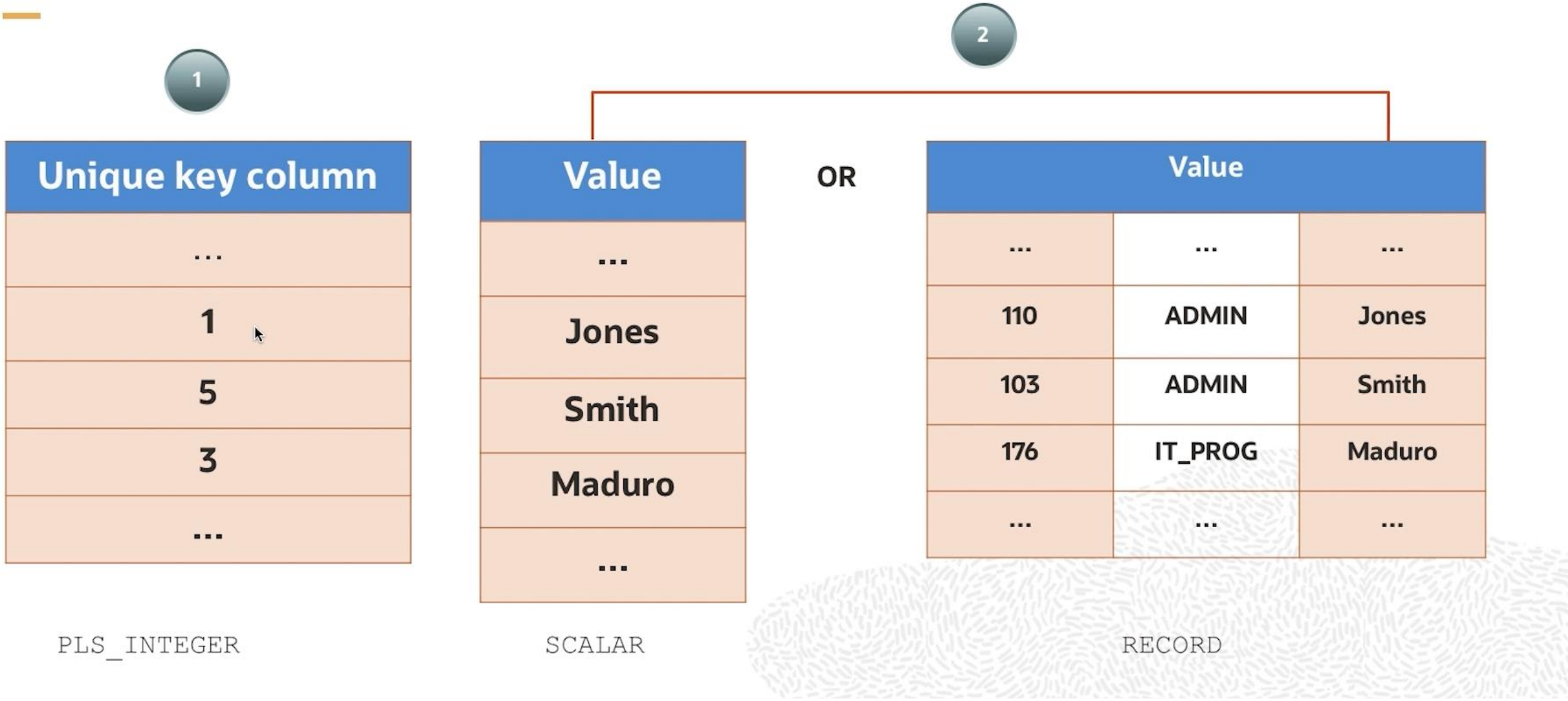
Associative Arrays (`INDEX BY` Tables)

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	Value
1	JONES
2	HARDEY
3	MADURO
4	KRAMER

Associative Array Structure



Steps to Create an Associative Array

Syntax:

```
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>) } ;
identifier    type_name;
```

Example:

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

Steps to Create an Associative Array

Syntax:

```
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>) } ;
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 email_table IS TABLE OF
    employees.email%TYPE
    INDEX BY PLS_INTEGER;
    email_list       email_table;
BEGIN
   email_list(100) = 'SKING';
   email_list(105) = 'DAUSTIN';
   email_list(110) = 'JCHEN';
   DBMS_OUTPUT.PUT_LINE(email_list(100));
   DBMS_OUTPUT.PUT_LINE(email_list(105));
   DBMS_OUTPUT.PUT_LINE(email_list(110));
END;
/
...
```

```
PL/SQL procedure successfully completed.

SKING
DAUSTIN
JCHEN
```

Associative Arrays with Record values

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;
/
```

```
PL/SQL procedure successfully completed.

10  Administration  200
```

Using Collection Methods

```
DECLARE
  TYPE email_table IS TABLE OF
    employees.email%TYPE
    INDEX BY PLS_INTEGER;
    email_list       email_table;
BEGIN
   email_list(100) = 'SKING';
   email_list(105) = 'DAUSTIN';
   email_list(110) = 'JCHEN';
   DBMS_OUTPUT.PUT_LINE('The number of elements in the list' ||
  email_list.COUNT);
   DBMS_OUTPUT.PUT_LINE('The first index in the list '||  email_list.FIRST);
   DBMS_OUTPUT.PUT_LINE('The last index in the list ' ||    email_list.LAST);
END;
/
```

Script Output

Task completed in 0.005 seconds

```
PL/SQL procedure successfully completed.

The number of elements in the list 3
The first index in the list 100
The last index in the list 110
```

Using Collection Methods with Associative Arrays

```
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;
/
```

```
PL/SQL procedure successfully completed.

King
Kochhar
De Haan
Hunold
Ernst
```

Quiz

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 how to:

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

Practice 7: Overview

This practice covers the following topics:

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