Working with Composite Data Types

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`

What is a PL/SQL Record

A PL/SQL record is a composite data structure that is a group of related data stored in *fields*.

Each field in the PL/SQL record has its own name and data type.

Declaring a PL/SQL Record

1- *programmer-defined records.*

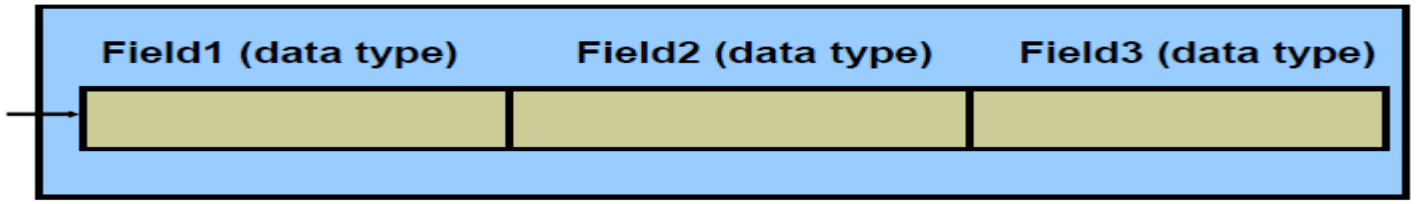
2- table-based record. %Rowtype

3- cursor-based record. (will be covered later)

PL/SQL Records
1- programmer-defined records

To declare programmer-defined record, first you have to define a record type by using TYPE statement with the fields of record explicitly.
Then, you can declare a record based on record type that you've defined.

```
DECLARE

TYPE t_EMP IS RECORD
( V_EMP_id employees.employee_id%type,
  v_first_name employees.first_name%type,
  v_last_name employees.last_name%type
);

v_emp t_EMP;

BEGIN
```



Continue ….
1- programmer-defined records

PL/SQL Records

A record is a group of related data items stored in fields, each with its own name and data type.

- Each record defined can have as many fields as necessary.
- Records can be assigned initial values and can be defined as `NOT NULL`.
- Fields without initial values are initialized to `NULL`.
- The `DEFAULT` keyword can also be used when defining fields.
- You can define `RECORD` types and declare user-defined records in the declarative part of any block, subprogram, or package.
- You can declare and reference nested records. One record can be the component of another record.

1- programmer-defined records

Example

```
DECLARE

TYPE t_EMP IS RECORD
( V_EMP_id employees.employee_id%type,
  v_first_name employees.first_name%type,
  v_last_name employees.last_name%type
);

v_emp t_EMP;

BEGIN
  select employee_id  ,first_name        ,last_name
  into v_emp
  from
  employees
  where employee_id=100;

  dbms_output.put_line(v_emp.V_EMP_id||' '||v_emp.v_first_name||' '||v_emp.v_last_name);

END;
```

PL/SQL Records
2- table-based record %Rowtype

%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;
```

```
DECLARE
 emp_record  employees%ROWTYPE;
 ...
```

PL/SQL Records
2- table-based record %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.

PL/SQL Records

2- table-based record %Rowtype

```
--using the %rowtype
declare

v_dept DEPARTMENTS%rowtype;

begin

select department_id,department_name,manager_id,location_id
into v_dept
from DEPARTMENTS  where department_id=10;

insert into copy_DEPARTMENTS values v_dept;
/*
insert into copy_DEPARTMENTS values (v_dept.department_id,v_dept.department_name,.....
*/

end;
```

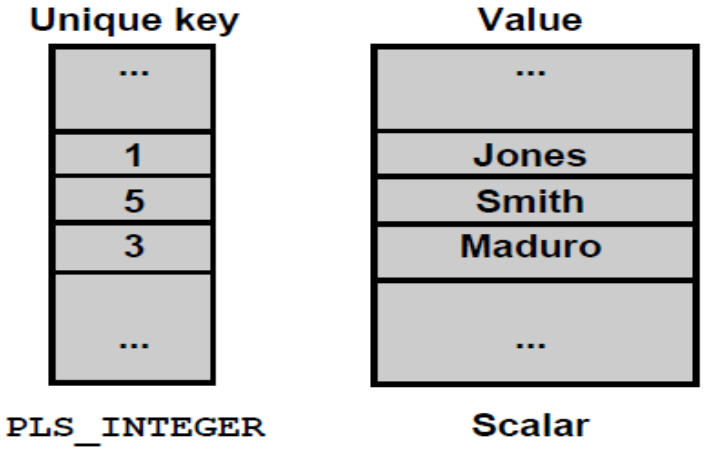
Composite Data Types

- 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`

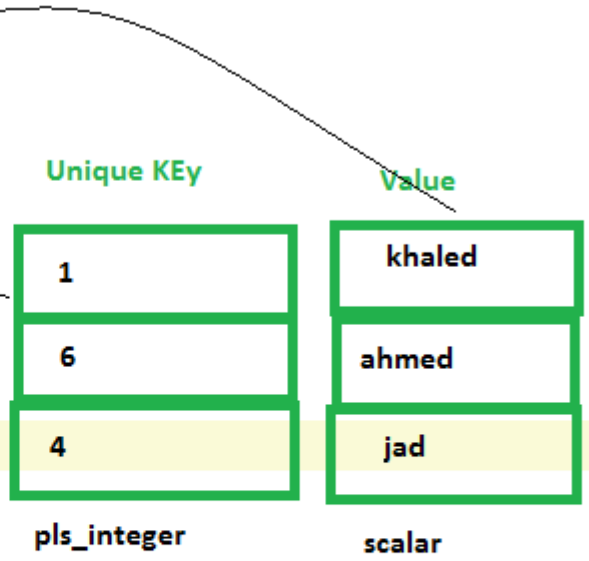
https://www.youtube.com/watch?v=m1Agc0XE0Po&list=PL0mkplxCP4yizj8BnM9MxJRxJqKvbmwXS

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.



```oracle
declare

type tab_no is table of varchar2(100)
index by pls_integer;

v_tab_no tab_no;

begin

v_tab_no(1):='khaled';
v_tab_no(6):='ahmed';
v_tab_no(4):='jad';

dbms_output.put_line(v_tab_no(1));
dbms_output.put_line(v_tab_no(6));
dbms_output.put_line(v_tab_no(4));
end;
```



Syntax:

```oracle
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;
```

Using INDEX BY Table Methods

The following methods make INDEX BY tables easier to use:

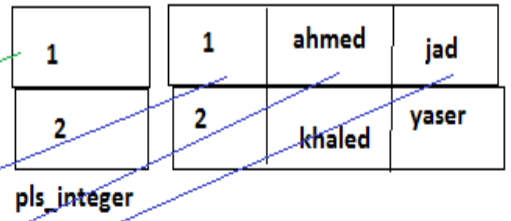
- EXISTS
- COUNT
- FIRST
- LAST
- PRIOR
- NEXT
- DELETE

Syntax: *table_name.method_name[(parameters)]*

Method	Description
EXISTS (*n*)	Returns TRUE if the *n*th element in a PL/SQL table exists
COUNT	Returns the number of elements that a PL/SQL table currently contains
FIRST	• Returns the first (smallest) index number in a PL/SQL table • Returns NULL if the PL/SQL table is empty
LAST	• Returns the last (largest) index number in a PL/SQL table • Returns NULL if the PL/SQL table is empty
PRIOR (*n*)	Returns the index number that precedes index *n* in a PL/SQL table
NEXT (*n*)	Returns the index number that succeeds index *n* in a PL/SQL table
DELETE	• DELETE removes all elements from a PL/SQL table. • DELETE (n) removes the *n*th element from a PL/SQL table. • DELETE (m, n) removes all elements in the range *m* ... *n* from a PL/SQL table.

INDEX BY Table of Records

```
declare
type tab_no is table of employees%rowtype
index by pls_integer;

v_tab_no tab_no;
v_total number;

begin
v_tab_no(1).employee_id:=1;
v_tab_no(1).first_name:='ahmed';
v_tab_no(1).last_name:='jad';

v_tab_no(2).employee_id:=2;
v_tab_no(2).first_name:='khaled';
v_tab_no(2).last_name:='yaser';

dbms_output.put_line(v_tab_no(1).employee_id||v_tab_no(1).first_name||v_tab_no(1).last_name);
dbms_output.put_line(v_tab_no(2).employee_id||v_tab_no(2).first_name||v_tab_no(2).last_name);

end;
```



Nested Tables

Example:

```
TYPE location_type IS TABLE OF locations.city%TYPE;
offices location_type;
```

- No index in nested table (unlike index by table)
- It is valid data type in SQL (unlike index by table, only used in PL/SQL)
- Initialization required
- Extend required
- Can be stored in DB

Syntax

```
TYPE type_name IS TABLE OF
 {column_type | variable%TYPE
 | table.column%TYPE} [NOT NULL]
 | table.%ROWTYPE
```

Nested Tables

```
declare
type t_locations is table of varchar2(100);

loc t_locations;

begin

loc:=t_locations('jordan','uae','Syria');

dbms_output.put_line(loc(1) );
dbms_output.put_line(loc(2) );
dbms_output.put_line(loc(3) );

end;
```

VARRAY

```
declare
type t_locations is varray(3) of varchar2(100);

loc t_locations;

begin

loc:=t_locations('jordan','uae','Syria');

dbms_output.put_line(loc(1) );
dbms_output.put_line(loc(2) );
dbms_output.put_line(loc(3) );

end;
```

characteristics for each type of collections

Collection Type	Number of Elements	Subscript Type	Dense or Sparse	Where Created	Can Be Object Type Attribute
Associative array (or index-by table)	Unbounded	String or integer	Either	Only in PL/SQL block	No
Nested table	Unbounded	Integer	Starts dense, can become sparse	Either in PL/SQL block or at schema level	Yes
Variable-size array (Varray)	Bounded	Integer	Always dense	Either in PL/SQL block or at schema level	Yes

Thank You