User Defined Data Types

Oracle 8 onwards user-defined data types can be created.

When pre-defined data types are not able to meet a business requirement, we create our own data types.

Types of User Defined Data Types

Composite Type / Object Type

Collections

- Associative array
- VARRAY
- Nested Table

Composite Type

Composite type allows group of elements of different types.

Syntax

create [or replace] type object_type_name as object

(field1, datatype, (size), field2, datatype, (size), ...));

USER-DEFINED DATATYPES OBJECT TYPE

Object for Multivalued Attributes

```
CREATE TYPE  address_type  AS OBJECT
(
street VARCHAR2(100),
city VARCHAR2(50),
state VARCHAR2(50),
zip_code VARCHAR2(10)
)
```

```
create table ppp
(
id number(4),
address address_type
);
```

```
insert into ppp values(101, address_type('Turner road', 'Dehradun', 'Uttrakhand', '248002'));
```

```
select * from ppp;
```

```
select id, t.address.street, t.address.street, t.address.CITY, t.address.STATE, t.address.ZIP_CODE from ppp t;
```

Varray for Multivalued Attributes

Syntax

```
create [or replace] type type_name is

varray(array_size) of datatype[(element_size)];
```

```
SQL> create type phone_array is varray(3) of number(10);
SQL> create table emp_ph (eid number(3), phone phone_array);
SQL> insert into emp_ph values(121, phone_array(9890, 9800));
```

Create a nested table using user-defined datatypes

First, we create the user-defined datatype

```
SQL> create type location as object
(city varchar2(30), PIN number(6)); /
```

```
SQL> create type nest_table as table of location; /
```

Now we will create the nested table

```
SQL> create table student
(name varchar2(10),
address nest_table)
nested table address
store as student_temp
/
```

Now we insert the record into the table

```
SQL> insert into student values
('mohit', nest_table(location('DDN', 248002)));
```

Insert the two locations for the same record

```
insert into student values
('Isha', nest_table(location('Pune',
400001),location('Lucknow', 248002)));
```

Display the record

```
SQL> select * from student;
```

We use the

```
SQL> select  name, s.city, s.pin from
student s1, table(s1.address) s;
```

Updation of the record in nested table

```
update table
(select address from student where name = 'Ravi') s
set
s.city = 'Varanasi' where s.city = 'DDN';
```

Package, Package Variables: Public Variables

A **package** is a schema object that groups logically related PL/SQL types, variables, constants, subprograms, cursors, and exceptions. A package is compiled and stored in the database, where many applications can share its contents. You can think of a package as an application.
A package always has a **specification**, which declares the **public items** that can be referenced from outside the package.

First, create the package

```
create or replace package pc1 as
n1 number;
n2 number;
n3 number;
v1 varchar2(100);
v2 varchar2(100);
end;
/
```

Next, use the package variables in a procedure

```
begin
pc1.n1 := 10;
pc1.n2 := 20;
pc1.n3 := pc1.n1+ pc1.n2;
dbms_output.put_line(pc1.n3);
end;
/
```

Create a package with a procedure and a function.

```
create or replace package pc1 as
procedure hello;
function addf(n1 number, n2 number)
return number;
end;
/
```

```
create or replace package body pc1 as
procedure hello as
begin
dbms_output.put_line('Hello World!');
end hello;
function addf(n1 number, n2 number)
return number is
begin
return n1 + n2;
end addf;
end;
/
```

Now execute the procedure using exec as follows

```
SQL> exec pc1.hello;
```

Now, execute the function using dual as follows

```
SQL> select pc1.addf(10, 20) from dual;
```

Create a package with a procedure that will be used to fetch employee details based on employee_id.

```
create or replace package pc2 is
procedure esal(eid IN number);
end;
/
```

```
create or replace package body pc2 is
procedure esal(eid IN number) is
e employees%rowtype;
begin
select * into e from employees
where employee_id=eid;
dbms_output.put_line(e.employee_id||' '
||e.first_name ||' '||e.salary);
exception
when no_data_found then
dbms_output.put_line('Sorry! Employee Not Existed. ');
end esal;
end;
/
```

Now we will execute the procedure as follows

SQL> exec pc2. esal(101);

Function Overloading

```
create package pc3 is
function f1(n1 in number, n2 in number)
return number;
function f1(n1 in number, n2 in number, n3 number)
return number;
end;
```

```
create or replace package body pc3 is
function f1(n1 in number, n2 in number)
return number is
begin
return n1 + n2;
end f1;
function f1(n1 in number, n2 in number,
n3 in number) return number is
begin
return n1 + n2 + n3;
end f1;
end;
/
```

Now, execute the function using dual as follows

```
SQL> select pc3.f1(10, 20) from dual;

SQL> select pc3.f1(10, 20, 30) from dual;
```

Procedure Overloading

```
create package pc4 is
procedure f1(n1 in number, n2 in number);
procedure f1(n1 in number, n2 in number,
n3 number);
end;
/
```

```
create package body pc4 is
procedure f1(n1 in number, n2 in number) is
begin
dbms_output.put_line(n1+ n2);
end f1;
procedure f1(n1 in number, n2 in number,
n3 in number) is
begin
dbms_output.put_line(n1+ n2+ n3);
end f1;
end;
/
```

Now, execute the function using exec as follows

```
SQL> exec pc4.f1(10, 20);

SQL> exec pc4.f1(10, 20, 30);
```

Another combination when procedure and function have the same name

```
create package pc5 is
procedure f1(n1 in number, n2 in number);
function f1(n1 in number, n2 in number)
return number;
end;
/
```

```
create package body pc5 is
procedure f1(n1 in number, n2 in number) is
begin
dbms_output.put_line(n1+ n2);
end f1;
function f1(n1 in number, n2 in number)
return number is
begin
return n1 + n2;
end f1;
end;
/
```

Now, execute the procedure using exec as follows

```
SQL> exec pc5.f1(10, 20);
```

Next, we will execute the function using dual as follows

```
SQL> select pc5.f1(10, 20) from dual;
```