PL/SQL - OBJECT ORIENTED

PL/SQL allows defining an object type, which helps in designing object-oriented database in Oracle. An object type allows you to crate composite types. Using objects allow you implementing real world objects with specific structure of data and methods for operating it. Objects have attributes and methods. Attributes are properties of an object and are used for storing an object's state; and methods are used for modeling its behaviors.

Objects are created using the CREATE [OR REPLACE] TYPE statement. Below is an example to create a simple **address** object consisting of few attributes:

```
CREATE OR REPLACE TYPE address AS OBJECT
(house_no varchar2(10),
 street varchar2(30),
 city varchar2(20),
 state varchar2(10),
 pincode varchar2(10)
);
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Type created.
```

Let's create one more object **customer** where we will wrap **attributes** and **methods** together to have object oriented feeling:

```
CREATE OR REPLACE TYPE customer AS OBJECT
(code number(5),
 name varchar2(30),
 contact_no varchar2(12),
 addr address,
 member procedure display
);
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Type created.
```

Instantiating an Object

Defining an object type provides a blueprint for the object. To use this object, you need to create instances of this object. You can access the attributes and methods of the object using the instance name and **the access operator .** as follows:

```
DECLARE
   residence address;
BEGIN
   residence := address('103A', 'M.G.Road', 'Jaipur', 'Rajasthan','201301');
   dbms_output.put_line('House No: '|| residence.house_no);
   dbms_output.put_line('Street: '|| residence.street);
   dbms_output.put_line('City: '|| residence.city);
   dbms_output.put_line('State: '|| residence.state);
   dbms_output.put_line('Pincode: '|| residence.pincode);
END;
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
House No: 103A
```

```
Street: M.G.Road
City: Jaipur
State: Rajasthan
Pincode: 201301

PL/SQL procedure successfully completed.
```

Member Methods

Member **methods** are used for manipulating the **attributes** of the object. You provide the declaration of a member method while declaring the object type. The object body defines the code for the member methods. The object body is created using the CREATE TYPE BODY statement.

Constructors are functions that return a new object as its value. Every object has a system defined constructor method. The name of the constructor is same as the object type. For example:

```
residence := address('103A', 'M.G.Road', 'Jaipur', 'Rajasthan','201301');
```

The **comparison methods** are used for comparing objects. There are two ways to compare objects:

- **Map method**: The **Map method** is a function implemented in such a way that its value depends upon the value of the attributes. For example, for a customer object, if the customer code is same for two customers, both customers could be the same and one. So the relationship between these two objects would depend upon the value of code.
- **Order method**: The **Order methods** implement some internal logic for comparing two objects. For example, for a rectangle object, a rectangle is bigger than another rectangle if both its sides are bigger.

Using Map method

Let us try to understand above concepts using the following rectangle object:

```
CREATE OR REPLACE TYPE rectangle AS OBJECT
(length number,
 width number,
 member function enlarge( inc number) return rectangle,
 member procedure display,
 map member function measure return number
);
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Type created.
```

Creating the type body:

```
CREATE OR REPLACE TYPE BODY rectangle AS
   MEMBER FUNCTION enlarge(inc number) return rectangle IS
   BEGIN
      return rectangle(self.length + inc, self.width + inc);
   END enlarge;

   MEMBER PROCEDURE display IS
   BEGIN
      dbms_output.put_line('Length: '|| length);
      dbms_output.put_line('Width: '|| width);
   END display;

   MAP MEMBER FUNCTION measure return number IS
   BEGIN
      return (sqrt(length*length + width*width));
```

```
   END measure;
END;
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Type body created.
```

Now using the rectangle object and its member functions:

```
DECLARE
   r1 rectangle;
   r2 rectangle;
   r3 rectangle;
   inc_factor number := 5;
BEGIN
   r1 := rectangle(3, 4);
   r2 := rectangle(5, 7);
   r3 := r1.enlarge(inc_factor);
   r3.display;

   IF (r1 > r2) THEN -- calling measure function
      r1.display;
   ELSE
      r2.display;
   END IF;
END;
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Length: 8
Width: 9
Length: 5
Width: 7

PL/SQL procedure successfully completed.
```

Using Order method

Now, the **same effect could be achieved using an order method**. Let us recreate the rectangle object using an order method:

```
CREATE OR REPLACE TYPE rectangle AS OBJECT
(length number,
 width number,
 member procedure display,
 order member function measure(r rectangle) return number
);
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Type created.
```

Creating the type body:

```
CREATE OR REPLACE TYPE BODY rectangle AS
   MEMBER PROCEDURE display IS
   BEGIN
      dbms_output.put_line('Length: '|| length);
      dbms_output.put_line('Width: '|| width);
   END display;

   ORDER MEMBER FUNCTION measure(r rectangle) return number IS
```

```
   BEGIN
      IF(sqrt(self.length*self.length + self.width*self.width)> sqrt(r.length*r.length +
r.width*r.width)) then
         return(1);
      ELSE
         return(-1);
      END IF;
   END measure;
END;
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Type body created.
```

Using the rectangle object and its member functions:

```
DECLARE
   r1 rectangle;
   r2 rectangle;
BEGIN
   r1 := rectangle(23, 44);
   r2 := rectangle(15, 17);
   r1.display;
   r2.display;
   IF (r1 > r2) THEN -- calling measure function
      r1.display;
   ELSE
      r2.display;
   END IF;
END;
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Length: 23
Width: 44
Length: 15
Width: 17
Length: 23
Width: 44

PL/SQL procedure successfully completed.
```

Inheritance for PL/SQL Objects:

PL/SQL allows creating object from existing base objects. To implement inheritance, the base objects should be declared as NOT FINAL. The default is FINAL.

The following programs illustrate inheritance in PL/SQL Objects. Let us create another object named TableTop, which is inheriting from the Rectangle object. Creating the base *rectangle* object:

```
CREATE OR REPLACE TYPE rectangle AS OBJECT
(length number,
 width number,
 member function enlarge( inc number) return rectangle,
 NOT FINAL member procedure display) NOT FINAL
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Type created.
```

Creating the base type body:

```
CREATE OR REPLACE TYPE BODY rectangle AS
   MEMBER FUNCTION enlarge(inc number) return rectangle IS
   BEGIN
      return rectangle(self.length + inc, self.width + inc);
   END enlarge;

   MEMBER PROCEDURE display IS
   BEGIN
      dbms_output.put_line('Length: '|| length);
      dbms_output.put_line('Width: '|| width);
   END display;
END;
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Type body created.
```

Creating the child object *tabletop*:

```
CREATE OR REPLACE TYPE tabletop UNDER rectangle
(
   material varchar2(20);
   OVERRIDING member procedure display
)
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Type created.
```

Creating the type body for the child object *tabletop*:

```
CREATE OR REPLACE TYPE BODY tabletop AS
OVERRIDING MEMBER PROCEDURE display IS
BEGIN
   dbms_output.put_line('Length: '|| length);
   dbms_output.put_line('Width: '|| width);
   dbms_output.put_line('Material: '|| material);
END display;
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Type body created.
```

Using the *tabletop* object and its member functions:

```
DECLARE
   t1 tabletop;
   t2 tabletop;
BEGIN
   t1:= tabletop(20, 10, 'Wood');
   t2 := tabletop(50, 30, 'Steel');
   t1.display;
   t2.display;
END;
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Length: 20
Width: 10
```

```
Material: Wood
Length: 50
Width: 30
Material: Steel

PL/SQL procedure successfully completed.
```

Abstract Objects in PL/SQL

The NOT INSTANTIABLE clause allows you to declare an abstract object. You cannot use an abstract object as it is; you will have to create a subtype or child type of such objects to use its functionalities.

For example,

```
CREATE OR REPLACE TYPE rectangle AS OBJECT
(length number,
 width number,
 NOT INSTANTIABLE NOT FINAL MEMBER PROCEDURE display)
 NOT INSTANTIABLE NOT FINAL
/
```

When the above code is executed at SQL prompt, it produces the following result:

```
Type created.
```