Parameter Modes in PL/SQL Subprograms

	Parameter Mode & Description
1	**IN** An IN parameter lets you pass a value to the subprogram. **It is a read-only parameter**. Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter. You can also initialize it to a default value; however, in that case, it is omitted from the subprogram call. **It is the default mode of parameter passing. Parameters are passed by reference.**
2	**OUT** An OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. **The actual parameter must be variable and it is passed by value**.
2	**IN OUT** An IN OUT parameter passes an initial value to a subprogram and returns an updated value to the caller. It can be assigned a value and its value can be read. The actual parameter corresponding to an IN OUT formal parameter must be a variable, not a constant or an expression. Formal parameter must be assigned a value. **Actual parameter is passed by value.**

IN & OUT Mode Example 1

This program finds the minimum of two values, here procedure takes two numbers using IN mode and returns their minimum using OUT parameters.

```
DECLARE
   a number;
   b number;
   c number;

PROCEDURE Min(x IN number, y IN number, z OUT number) IS
BEGIN
   IF x < y THEN
      z:= x;
   ELSE
      z:= y;
   END IF;
END;

BEGIN
   a:= 33;
   b:= 57;
   Min(a, b, c);
   dbms_output.put_line(' Minimum of (33, 57): ' || c);
END;
/
```

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

```
 Minimum of (33, 57): 33

PL/SQL procedure successfully completed.
```

IN & OUT Mode Example 2

This procedure computes the square of value of a passed value. This example shows how we can use same parameter to accept a value and then return another result.

```
DECLARE
   a number;
PROCEDURE square(x IN OUT number) IS
BEGIN
  x := x * x;
END;
BEGIN
   a:= 23;
   square(a);
   dbms_output.put_line(' Square of (23): ' || a);
END;
/
```

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

```
Square of (23): 529

PL/SQL procedure successfully completed.
```

Methods for Passing Parameters

Actual parameters could be passed in three ways:

- Positional notation
- Named notation

POSITIONAL NOTATION

In positional notation, you can call the procedure as:

```
findMin(a, b, c, d);
```

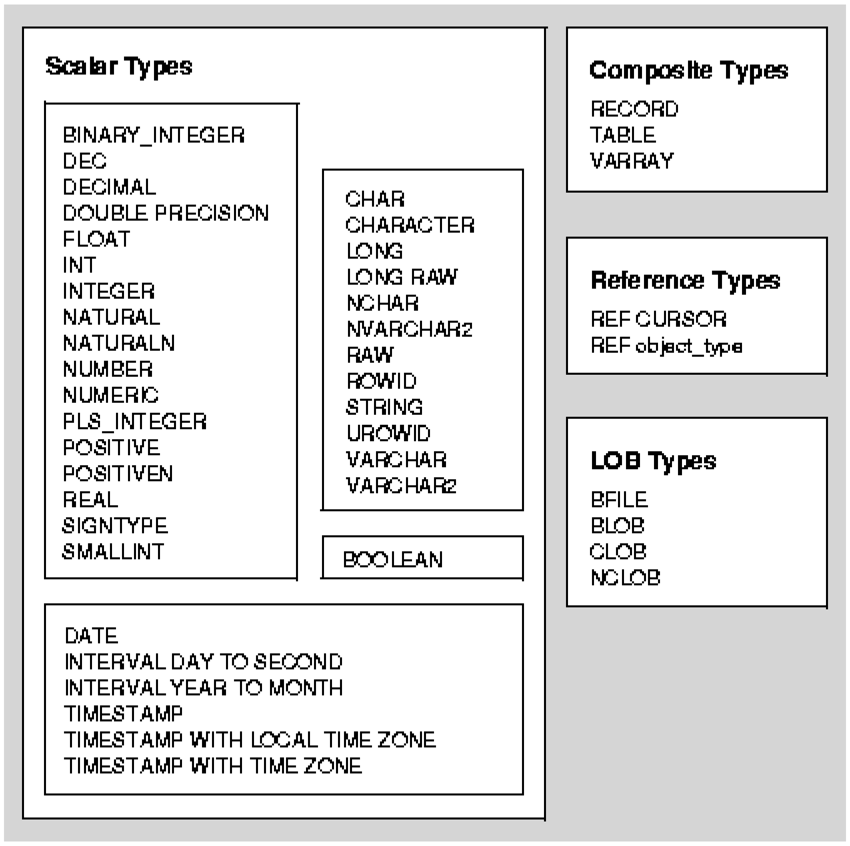
In positional notation, the first actual parameter is substituted for the first formal parameter; the second actual parameter is substituted for the second formal parameter, and so on. So, a is substituted for x, b is substituted for y, c is substituted for z and d is substituted for m.

NAMED NOTATION

In named notation, the actual parameter is associated with the formal parameter using the arrow symbol (=>). So the procedure call would look like:

```
findMin(x=>a, y=>b, z=>c, m=>d);
```

IN/OUT and IN/OUT can be any supported data types:



We do not support composite types as they are object types. Only the scalar data types are currently supported in ROracle + LOB. We like to support REF cursor with the procedures as it is just like a SELECT statement.

Here are some use cases:

1. Anonymous procedure with IN/OUT arguments:

   ```
   /* Create an anonymous stored procedure */

   dbGetQuery(con,  “CREATE PROCEDURE test
     (
       input IN INTEGER,
       output OUT INTEGER,
     )
     AS
     BEGIN
       select value into output
       FROM temp
       WHERE temp.id = input;
     END;”)


   /* Now execute the procedure */
   dbGetQuery(con, ' BEGIN test(:input, :output);   END;', data.frame(X = 1, Y))
   ```

2. Anonymous function returning a value

   ```
   dbGetQuery(con, "CREATE OR REPLACE FUNCTION TestFunc
               RETURN number IS
                 total number(2) := 0;
               BEGIN
                 SELECT count(*) into total
                 FROM emp;
                 RETURN total;
               END;")

   X <- 2
   dbGetQuery(con, 'BEGIN :1 := TestFunc(); END;', data.frame(X))
   ```

3. REF cursor example:

   ```
   dbGetQuery(con, "create or replace function
                get_dept_emps(p_deptno in number) return sys_refcursor is
   ```

```
          v_rc sys_refcursor;
  begin
    open v_rc for 'select empno, ename, mgr, sal from emp where deptno = :deptno' using
p_deptno;
    return v_rc;
  end;")

Y <- 2
X <- data.frame(NULL)
dbGetQuery(con, "BEGIN :1 := get_dept_emps(:2); END;", data.frame(X, Y))
```

Basically, we will allow any scalar value to be passed as an IN or OUT.