Functions and Procedures: Details

Dr. Abu Raihan Mostofa Kamal

December 15, 2021

Functions and Procedures: Parameters

- **IN:** The value of the actual parameter is passed into the procedure when the procedure is invoked. Inside the procedure, the formal parameter acts like a PL/SQL constant—it is considered **read-only** and cannot be changed.
- **OUT:** Any value the actual parameter has when the procedure is called is ignored. Inside the procedure, *the formal parameter acts like an uninitialized PL/SQL variable and thus has a value of NULL*. It can be read from and written to. **Local variables** are normally used to receive (through assignment operator:=) the value of OUT parameters.
- **INOUT:** This mode is a combination of IN and OUT.

Functions and Procedures: Parameters

- **IN:** The value of the actual parameter is passed into the procedure when the procedure is invoked. Inside the procedure, the formal parameter acts like a PL/SQL constant—it is considered **read-only** and cannot be changed.
- **OUT:** Any value the actual parameter has when the procedure is called is ignored. Inside the procedure, *the formal parameter acts like an uninitialized PL/SQL variable and thus has a value of NULL*. It can be read from and written to. **Local variables** are normally used to receive (through assignment operator:=) the value of OUT parameters.
- **INOUT:** This mode is a combination of IN and OUT.

Functions and Procedures: Parameters

- **IN:** The value of the actual parameter is passed into the procedure when the procedure is invoked. Inside the procedure, the formal parameter acts like a PL/SQL constant—it is considered **read-only** and cannot be changed.
- **OUT:** Any value the actual parameter has when the procedure is called is ignored. Inside the procedure, *the formal parameter acts like an uninitialized PL/SQL variable and thus has a value of NULL*. It can be read from and written to. **Local variables** are normally used to receive (through assignment operator:=) the value of OUT parameters.
- **INOUT:** This mode is a combination of IN and OUT.

IN Parameter: Example

```
1  CREATE FUNCTION DOIT(P_ID IN NUMBER)
2  RETURN NUMBER
3
4  IS
5
6  BEGIN
7  --it can not be written to--
8  P_ID:=12;
9
10 RETURN 1;
11 END;
12
```

OUT Parameter: Example

```
1   CREATE PROCEDURE DOIT(P_ID IN
    NUMBER, P_X OUT NUMBER)
2
3   IS
4
5   BEGIN
6
7   P_X := P_ID * P_ID;
8
9
10  END;
11
12
13
```

```
1
2   --Call from annonymous block
3   DECLARE
4   P number:=12;
5   X number;
6   BEGIN
7   DOIT(P,X);
8
9   dbms_output.put_line('The
    value of out pm X is : ' || X);
10
11  END;
12
13  --OUTPUT--
14  The value of out pm X is : 144
15
```

INOUT Parameter: Example

```
1  CREATE PROCEDURE DOIT(P_ID INOUT
   NUMBER)
2
3  IS
4
5  BEGIN
6
7  P_ID := P_ID*P_ID;
8
9
10
11 END;
12
13
14
```

```
1
2  --Call from annonymous block
3  DECLARE
4  X:=12;
5  BEGIN
6  DOIT(X);
7
8  dbms_output.put_line('Changed
   value of X is : ' || X);
9
10 END;
11
12 --OUTPUT--
13 Changed value of X is : 144
14
```

OUT Parameter: Applications

Why OUT parameter?

Complex processing requires **a number of return values**. Example, Banking, Law Agency, Airline Automation with Passengers status. Instead of using a number of traditional functions, a single function with multiple OUT parameters can be used.

Datatype Indicator

- Using the `%TYPE Attribute` Single Variable
- Using the `%ROWTYPE Attribute` Structure

Note: Constraints are **not inherited** from the base table. It only copies the name and data type from the referenced table.

Datatype Indicator by Example

```
1  CREATE TABLE EMP(ID NUMBER(4,0) PRIMARY KEY,
2  DEPT NUMBER(2,0) DEFAULT 20,
3  CONSTRAINT CHK CHECK (DEPT BETWEEN 20 and 40)
4  );
5
```

Datatype Indicator by Example

```
1
2   DECLARE
3   emprec    employees_temp%ROWTYPE;
4   id emp.id%TYPE;
5   BEGIN
6   emprec.id := NULL;  -- this works, null constraint is not inherited
7   -- emprec.id := 10000002; -- invalid, number precision too large
8   emprec.dept := 50; -- this works, check constraint is not inherited
9   -- the default value is not inherited in the following
10  DBMS_OUTPUT.PUT_LINE('emprec.dept: ' || emprec.dept);
11  END;
12  /
13
```

Datatype Indicator: Benefits

- It is a **short-cut** to copy and past in variable declaration
- It is **dynamic**, since base table data type change is immediately reflected here without any modification of the underlying code.

Parameter Positions

1. **Positional Notation:** You use positional notation to call the function as follows:

```
BEGIN
dbms_output.put_line(add_three_numbers(3,4,5));
END;
```

2. **Named Notation:** You call the function using named notation by:

```
BEGIN
dbms_output.put_line(add_three_numbers(c => 4,b => 5,c => 3));
END;
```

Parameter Positions

4. **Mixed Notation:** You call the function by a mix of both positional and named notation by:

```
BEGIN
dbms_output.put_line(add_three_numbers(3,c => 4,b => 5));
END;
```

Note: There is a restriction on mixed notation. All positional notation actual parameters must occur first and in the same order as they are defined by the function signature.

Calling Procedure and Function

Executing a Standalone Procedure: For procedure you cannot call as part of your SELECT statement (since it does not return anything). It can be called in 2 ways:

1. Using the **EXECUTE** keyword
2. Calling the name of the procedure from a PL/SQL block

```
1 --from SQL terminal
2 EXECUTE apply_discount(12,.12);
3
4 ---wihing a block
5   BEGIN
6   apply_discount( new_company_id, 0.15 ); -- 15% discount
7   END;
```

Calling Procedure and Function(Cont.)

Function: You can call it in any SELECT statement or in using any local variable inside a block.

```
1
2   SELECT ID, NAME, GET_GPA(ID) RESULT
3   FROM STUDENTS;
4
5   ---from a block
6   s number:=0;
7   .....
8
9   s:=GET_GPA(101);
10
11
```

Have a look on different Built-in Functions and RE

List of most frequently used functions:

```
1
2     LOWER, UPPER, RPAD, LPAD, LTRIM, LTRIM, LENGTH
3     SUBSTR, INSTR, DMBS_RANDOM.VALUE(x,y)
4
5
```

SUBSTR Example, n1=startpoint (-ve from reverse), n2=len of substr

```
SUBSTR('This is a test', 6, 2)
Result: 'is'

SUBSTR('This is a test', 6)
Result: 'is a test'

SUBSTR('TechOnTheNet', 1, 4)
Result: 'Tech'

SUBSTR('TechOnTheNet', -3, 3)
Result: 'Net'

SUBSTR('TechOnTheNet', -6, 3)
Result: 'The'
```

Regular Expression

Regular expressions enable you to search for patterns in string data by using standardized syntax conventions. Lets learn by few **examples**.
Remember some **Metacharacters**:

```
[] signle character
 [A-Z] a single CAPILTAL letter
 [A-Z]+ one or more
 \d{n} n digits
```

NOT YET FINISHED