PL/SQL (Elective - I)

Unit - 4

PL/SQL Subprograms

Subprogram: A subprogram is a logical group of SQL and PL/SQL statements that perform a specific task. These sub-programs are combined to form larger programs. Sub-programs support the concept of modular programming where a large and complex program is decomposed into multiple smaller modules or sub-programs.

PL/SQL subprograms can be created as procedure or function. These sub-programs can be called by passing a set of parameters. A PL/SQL subprogram (either procedure or function) is made of 3 parts and they are:-

i.) **Declarative Part:**

The declarative part may contain the declaration of variables, constants, cursors and other sub program. This part is optional and included only if necessary.

ii.) **Executable Part:-**

This is the ~~compulsory~~ compulsory part and contains all SQL and PL/SQL statements necessary to perform desired task.

iii.) **Exception Handling Part (optional):-**

This is an optional part and it contains the code that handles run-time errors (exceptions) of

Scanned with CamScanner

executable part.

Procedure:

A procedure is a block of PL/SQL statement that does not return a value directly. However, a procedure may accept 0 or any number of arguments. It is designed to perform a specific task. A procedure is a small unit of program that can perform a specific task and can be executed independently.

Scanned with CamScanner

PL/SQL

Procedure :-

Syntax : CREATE OR REPLACE PROCEDURE procedure name
(Argument {IN, OUT, IN OUT} Datatype, ...)
[Is, As]
Variable declarations;
.....
.....
BEGIN
PL/SQL Subprogram body;
EXCEPTION
PL/SQL exception handling body;
END;

Where,
REPLACE recreates the procedure if already exist.
IN - IN indicates that the parameter will accept a value from the user.
OUT - OUT indicates that the parameter will return a value to the user
IN OUT - IN OUT indicates that the parameter will either accept a value or return a value.

for example :-

Scanned with CamScanner

A simple procedure to print the message Hello Student

```
CREATE OR REPLACE PROCEDURE hello
IS
BEGIN
    dbms_output.put_line('Hello Students');
END;
```

A procedure can be called in two different ways as:-

i) Using the EXECUTE keyword

For Eg:- EXECUTE hello;

ii) Calling the name of the procedure from another PL/SQL block.

```
BEGIN
    hello;
END;
```

*. Write a procedure that takes two numbers as input and finds the greater number.

Scanned with CamScanner

```
CREATE OR REPLACE PROCEDURE greater (x IN
                                    NUMBER,
        y IN NUMBER, z OUT NUMBER)
IS
  BEGIN
    IF x > y THEN
      z := x;
    ELSE
      z := y;
    END IF;
  END;
```

Now, the PL/SQL code block to call the procedure will be as:-

```
DECLARE
  a NUMBER; b NUMBER; C NUMBER;
BEGIN
    a := :a;
    b := :b;
  greater (a, b, c);
   dbms_output.put_line ('Greater No is' || c);
END;
```

Scanned with CamScanner

*. Write a procedure to calculate the square of any number also write the PL/SQL code block.

```
CREATE OR REPLACE PROCEDURE square (x IN OUT
                                    NUMBER)
Is
BEGIN
    x := x * x;
END;
```

Now, the PL/SQL code block to call this procedure,

```
DECLARE
    a NUMBER;
BEGIN
    a := :a;
    square (a);
dbms_output.put_line ('Square = ' || a);
END;
```

Scanned with CamScanner

PL/SQL Elective-I

Function :-

A function in PL/SQL is same as procedure except that it returns a value to the calling block. It is mainly used to accept arguments, to perform certain calculation and return a single value at a time.

A function is created as:-

```
CREATE OR REPLACE FUNCTION functionName
(argument IN Datatype,.....)
RETURN Datatype
[IS,AS]
   Variable declarations;
BEGIN
   PL/SQL function body;
EXCEPTION
   PL/SQL Exception Handling block.
END;
```

Here, RETURN specifies the type of data returned by the function.

for eg:-

A simple function to count the number of employees ~~in sales department.~~ who are programmer.

Scanned with CamScanner

```
CREATE OR REPLACE FUNCTION totalEmployees
RETURN NUMBER
Is
  total NUMBER;
  BEGIN
    SELECT COUNT (EmpNo) INTO total,
    FROM Emp WHERE post='Programmer';
    RETURN total;
END;
```

A simple PL/SQL block to call this function will be as:-

```
DECLARE
    t NUMBER;
BEGIN
    t := totalEmployees ( );
  dbms_output.put_line ('Total no. of Programmers are = '||t);
END;
```

Write a function that takes post as input arguments finds out the total number of employees in that post and return it and also write appropriate PL/SQL block to call this function.

Scanned with CamScanner

```
CREATE OR REPLACE FUNCTION totalEmployees
                    (pst IN VARCHAR(20)
RETURN NUMBER
IS
  total NUMBER;
BEGIN
   SELECT COUNT(EmpNo) INTO total
   FROM Emp WHERE post=pst;
   RETURN total;
END;
```

A simple PL/SQL block to call this function:-

```
DECLARE
 p VARCHAR(20);
 t NUMBER;
BEGIN
   p:=:p;
   t := totalEmployees(p);
dbms_output.put_line('total no. of Employees in '||
                    p||'are='||t);
END;
```

Scanned with CamScanner

* Define a function that accepts two number as arguments and returns the greater number. Also write PL/SQL block that takes two number as input and display the greater number using this number.

```
CREATE OR REPLACE FUNCTION greater (
      x IN NUMBER, y IN NUMBER)
 RETURN NUMBER
 IS
   z NUMBER;
 BEGIN
    IF x>y THEN
      z:=x;
    ELSE
      z:=y;
    END IF;
    RETURN z;
 END;
```

Now a PL/SQL block to call function

```
DECLARE
      a NUMBER;
      b NUMBER;
      c NUMBER;
BEGIN
    a:=a;
    b:=b;
    c:= greater(a, b);
```

Scanned with CamScanner

Elective- I

* Define a function to calculate a factorial value of any number and write PL/SQL block to call that function.

```
CREATE OR REPLACE FUNCTION fact (
        n IN NUMBER)
RETURN NUMBER
IS
   f NUMBER
BEGIN
    f := 1;
    FOR i IN 1 ... n
     f := f * i;
    END FOR;
     RETURN f;
   END;
```

Now, a PL/SQL block to call function,

```
DECLARE
    a NUMBER;
    b NUMBER;
BEGIN
    a:=a;
    b := fact (a);
  dbms_output.put_line ('Factorial value = ' || b);
END;
```

Scanned with CamScanner

Passing Parameters :-

(a) In PL/SQL we can pass parameters in 3 different ways while calling a function or procedure.

(i) Positional Method :- In positional method actual parameters used in function or procedure call must be in same order in function or procedure definition. They must have identical data types. We cannot change the relative order of actual or formal arguments. For example :-

```
CREATE FUNCTION fun(x NUMBER, y NUMBER,
                                z NUMBER)
----- --
=== ==
-- - --

END;
Function calling body
   BEGIN
     ----
     fun(b, c, a);
     ----
   END;
```

In this case, values of the actual arguments b, c, and a are copied to the formal arguments x, y and z in the same order.

Scanned with CamScanner

Named Method:-

In this method we use an arrow operator (=>) while calling the function or procedure to pass the parameters. In this case there is no fixed positions to pass the parameters. for example:-

```
CREATE FUNCTION fun (x NUMBER, y NUMBER,
                     z NUMBER).
.....
.....
END;
 Function calling body
  BEGIN
  ....
  fun (z => a, y => b, x => c);
  .....
  END;
```

Mixed Method:-

In this approach, few parameters are passed as positional and few parameters using named method. for example,-

```
CREATE FUNCTION fun (x NUMBER, y NUMBER,
                     z NUMBER)
.....
.....
END;
```

Scanned with CamScanner

Function calling body

```
BEGIN
....
fun (a, z => b, y => c); .... legal call
fun (z => b, y => a, c); .... Illegal call
....
END;
```

In mixed method, parameters on left of function call must be positional. So we cannot write initial parameters as named and remaining parameters as positional.

Scanned with CamScanner

PL-SQL

Page No.
Date: / /

Advantages of Sub-programs

(a) Security: Procedures and functions can help to inforce data security. By providing permissions to different users to use the sub-program according their role we can control their use.

(b) Performance: It improves database performance in different ways as:

- Amount of information sent over the network is less.
- No complicated steps are required to execute the code.
- Once the sub-program is executed and result is available on disk, users can easily access the information available on disk.

(c) Memory allocation: The amount of memory required is reduced by the use of sub-program because sub-programs are shareable objects. Only one copy of sub-program is loaded into the memory for the execution by multiple users.

Scanned with CamScanner

(d.) Productivity : By writing procedures and functions, redundant coding can be avoided and it increases the productivity of programmers.

(e.) Integrity :- (Accuracy + Consistency) : It is easy to maintain data integrity using functions and procedures because sub-programs are only applied after testing their validity.

Scanned with CamScanner