

**Database Management System Lab(CS 29006)**

# **KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY**

**School of Computer Engineering**



Strictly for internal circulation (within KIIT) and reference only. Not for outside circulation without permission

***1 Credit***

**Course Committee**

# Lab Contents



2

Sr #	Major and Detailed Coverage Area	Lab#
1	<p>PL/SQL Programming Language</p> <ul style="list-style-type: none"><li><input type="checkbox"/> Procedure</li><li><input type="checkbox"/> Function</li></ul>	9

# Subprogram



3

A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the **calling program**.

PL/SQL provides two kinds of subprograms:

- ☐ **Functions:** these subprograms return a single value, mainly used to compute and return a value.
- ☐ **Procedures:** these subprograms do not return a value directly, mainly used to perform an action.

Procedures and Functions are saved in the database as **database objects**.

## *Parts of a PL/SQL Subprogram*

Like anonymous PL/SQL blocks and, the named blocks a subprograms will also have following three parts:

- ☐ Declarative Part
- ☐ Exception-handling Part
- ☐ Executable Part

# Terminologies in PL/SQL Subprograms



4

## Parameter:

The parameter is variable or placeholder of any valid PL/SQL datatype through which the PL/SQL subprogram exchange the values with the main code. This parameter allows to give input to the subprograms and to extract from these subprograms.

- ☐ These parameters should be defined along with the subprograms at the time of creation.
- ☐ These parameters are included in the calling statement of these subprograms to interact the values with the subprograms.
- ☐ The datatype of the parameter in the subprogram and in the calling statement should be same.
- ☐ The size of the datatype should not mention at the time of parameter declaration, as the size is dynamic for this type.

Based on their purpose parameters are classified as: **IN**, **OUT** and **IN OUT**

# Terminologies cont...



5

## IN:

- ☐ This parameter is used for giving input to the subprograms.
- ☐ It is a read-only variable inside the subprograms, their values cannot be changed inside the subprogram.
- ☐ In the calling statement these parameters can be a variable or a literal value or an expression, for example, it could be the arithmetic expression like '5\*8' or 'a/b' where 'a' and 'b' are variables.
- ☐ By default, the parameters are of IN type.

## OUT:

- ☐ This parameter is used for getting output from the subprograms.
- ☐ It is a read-write variable inside the subprograms, their values can be changed inside the subprograms.
- ☐ In the calling statement, these parameters should always be a variable to hold the value from the current subprograms.

# Terminologies cont...



6

## IN OUT:

- ❑ This parameter is used for both giving input and for getting output from the subprograms.
- ❑ It is a read-write variable inside the subprograms, their values can be changed inside the subprograms.
- ❑ In the calling statement, these parameters should always be a variable to hold the value from the subprograms.

## RETURN:

RETURN is the keyword that actually switch the control from the subprogram to the calling statement. In subprogram RETURN simply means that the control needs to exit from the subprogram. Once the RETURN keyword is encountered in the subprogram, the code after this will be skipped. Normally, parent or main block will call the subprograms, and then the control will shift from those parent block to the called subprograms. RETURN in the subprogram will return the control back to their parent block. In the case of functions RETURN statement also returns the value. The datatype of this value is always mentioned at the time of function declaration. The datatype can be of any valid PL/SQL data type.

# Procedure



7

## Syntax:

```
CREATE [OR REPLACE] PROCEDURE procedure_name [(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
    < procedure_body >
EXCEPTION
    < exception handling part >
END;
```

## Explanation:

- ☐ CREATE PROCEDURE is to create a new procedure. Keyword 'OR REPLACE' instructs is to replace the existing procedure (if any) with the current one.
- ☐ Procedure name should be unique.
- ☐ The optional parameter list contains name, mode and types of the parameters.
- ☐ Keyword '**IS**' will be used, when the procedure is nested into some other blocks. If the procedure is standalone then '**AS**' will be used.
- ☐ Procedure-body contains the executable part.
- ☐ Exception handling part contains the exception handling part.

# Standalone Procedure Example



8

```
CREATE OR REPLACE PROCEDURE Greetings
AS
BEGIN
    dbms_output.put_line('Hello World!');
END;
/
```

## *Executing the Standalone Procedure*

A standalone procedure can be called in two ways:

1. Using the EXECUTE keyword as shown below  
**EXECUTE Greetings;**
2. Calling the name of the procedure from a PL/SQL block as shown below  
**BEGIN**  
    **Greetings;**  
**END;**  
**/**



# Deleting Procedure



9

A standalone procedure is deleted with the **DROP PROCEDURE** statement. Syntax for deleting a procedure is: **DROP PROCEDURE procedure-name;**

So you can drop greetings procedure by using the following statement: **DROP PROCEDURE Greetings;**

## *IN & OUT mode example*

DECLARE

a,b,c number;

PROCEDURE findMin(x IN number, y IN number,  
z OUT number) IS

BEGIN

IF x < y THEN

z:= x;

ELSE

z:= y;

END IF;

END;

/\*continuation of program \*/

BEGIN

a:= 23;

b:= 45;

findMin(a, b, c);

dbms\_output.put\_line(' Minimum of (23, 45) : ' || c);

END;

/

# Nested Procedure Example cont...



10

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

# Methods for Passing Parameters



11

Assume there is an procedure `findMin(a, b, c, d)`. Actual parameters could be passed in three ways:

- ❑ **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 e.g. call the procedure as: `findMin(m, n, o, p)`;
- ❑ **Named notation:** the actual parameter is associated with the formal parameter using the arrow symbol ( `=>` ). So the procedure call would look like: `findMin(a=>m, b=>n, c=>o, d=>p)`;
- ❑ **Mixed notation:** In mixed notation, you can mix both notations in procedure call; however, the positional notation should precede the named notation.

This call is legal: `findMin(m, n, o, d=>p)`;

But this is not legal: `findMin(a=>x, b, c, d)`;

# Function



12

A PL/SQL function is same as a procedure except that it returns a value. Therefore, all the discussions of the previous chapter are true for functions too.

## **Creating Function Syntax:**

```
CREATE [OR REPLACE] FUNCTION function_name  
[(parameter_name [IN | OUT | IN OUT] type [, ...])]  
RETURN return_datatype  
{IS | AS}  
BEGIN  
    < function_body >  
EXCEPTION  
    < exception handling part >  
END;
```

# Standalone Function Example



13

```
CREATE OR REPLACE FUNCTION GetTotalCustomers
RETURN number
IS
    total number(2) := 0;
BEGIN
    SELECT count(*) into total FROM customer;
    RETURN total;
END;
/
```

*View structure with data*

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

# Calling a Function



14

To call a function you simply need to pass the required parameters along with function name and if function returns a value then you can store returned value. Following program calls the function **GetTotalCustomers** from an anonymous block:

```
DECLARE
```

```
  c number(2);
```

```
BEGIN
```

```
  c := GetTotalCustomers();
```

```
  dbms_output.put_line('Total no. of Customers: ' || c);
```

```
END;
```

```
/
```

# Nested Function Example



15

```
DECLARE
```

```
  a,b,c number;
```

```
FUNCTION findMax(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;
```

```
/* continuation of program */
```

```
BEGIN
```

```
  a:= 23;
```

```
  b:= 45;
```

```
  c := findMax(a, b);
```

```
  dbms_output.put_line(' Maximum of (23,45): ' || c);
```

```
END;
```

```
/
```

# PL/SQL Recursive Function

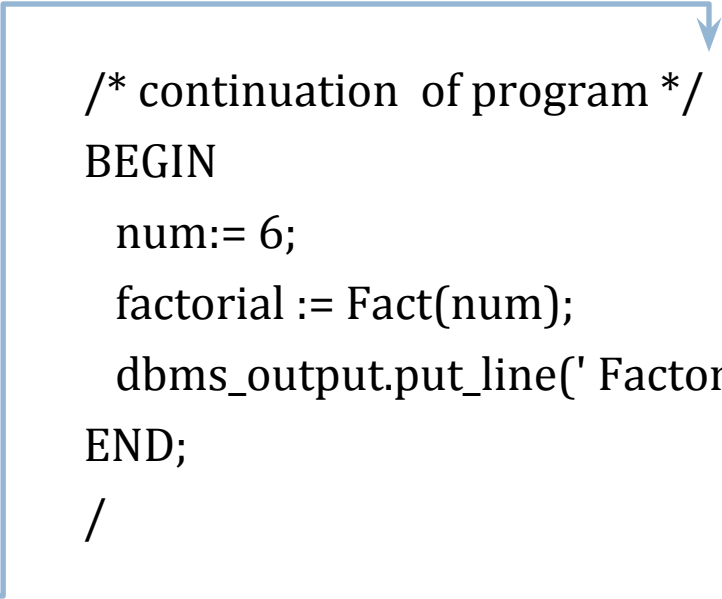


16

When a subprogram calls itself, it is referred to as a recursive call and the process is known as recursion. The following program calculates the factorial of a given number by calling itself recursively:

```
DECLARE
    num number;
    factorial number;

FUNCTION Fact(x number)
RETURN number
IS
    f number;
BEGIN
    IF x=0 THEN
        f := 1;
    ELSE
        f := x * Fact(x-1);
    END IF;
RETURN f;
END;
```

A blue line originates from the 'END;' statement of the 'Fact' function. It extends horizontally to the right, then turns vertically upwards, and finally turns horizontally to the right again, ending with a blue arrowhead pointing to the '/\* continuation of program \*/' comment. This diagram illustrates the recursive call mechanism where the function calls itself.

```
/* continuation of program */
BEGIN
    num:= 6;
    factorial := Fact(num);
    dbms_output.put_line(' Factorial ' || num || ' is ' || factorial);
END;
/
```



# Deleting Function



17

Once you have created your function, you might find that you need to remove it from the database.

**Syntax:** DROP FUNCTION function\_name;

**Example:** DROP FUNCTION GetTotalCustomers;  
DROP FUNCTION Fact;

# Difference between Function and Procedure



18

Procedure	Function
Used mainly to execute certain process	Used mainly to perform some calculation
Use OUT parameter to return the value	Use RETURN to return the value
It is not mandatory to return the value	It is mandatory to return the value
RETURN will simply exit the control from subprogram.	RETURN will exit the control from subprogram and also returns the value
Return datatype is not specified at the time of creation	Return datatype is mandatory at the time of creation



# Thank You

# End of Lab 9

# Assignment



20

## Reference Tables

- ❑ **EMPLOYEE** table with the attributes:  
ID, LAST\_NAME, FIRST\_NAME, MIDDLE\_NAME, FATHER\_NAME,  
MOTHER\_NAME, SEX, HIRE\_DATE, ADDRESS, CITY, STATE, ZIP, PHONE,  
PAGER, SUPERVISOR\_ID, DOB, INJECTED\_DATE
- ❑ **SCHOOL** table with the attributes:  
ID, NAME, INJECTED\_DATE
- ❑ **EMPLOYEE\_ALIGNMENT** table with the attributes:  
EMPLOYEE\_ID, SCHOOL\_ID, INJECTED\_DATE
- ❑ **JOB** table with the attributes:  
ID, NAME, TITLE, SALARY, BONUS , INJECTED\_DATE
- ❑ **EMPLOYEE\_PAY** table with the attributes:  
EMPLOYEE\_ID, JOB\_ID, INJECTED\_DATE

# Experiment



21

1. Create a PL/SQL procedure that outputs the message “I am a PL/SQL expert.”
2. Create a PL/SQL function to find out if a year is a leap year.
3. Create a PL/SQL procedure that takes employee ID as the input and displays the employee full name with name of the school.
4. Create a PL/SQL function that takes school ID as the input. If the school does not contain school name, return a false, otherwise return a true value. Print the appropriate message in the calling program based on the result.
5. Create a PL/SQL function to revise the salary by 20%, who works in the same school in which “John Smith” works? Print the appropriate message in the calling program based on the result.
6. Create a PL/SQL procedure that will present the analysis of the employee.
  - a. Birthdays of number of employees per each month
  - b. Ratio of male to female
  - c. Number of hires per each month
  - d. Number of employees working for each school
  - e. Number of employees with different job titles working for each school