**PROCEDURES**

# Advantages of Subprograms

* Provide Extensibility
  + PL/SQL language can be tailored to suit the needs of the application
* Promote reusability and maintainability
  + Once validated, they can be used with confidence in any number of applications
  + Simplifies maintenance/enhancement, as subprogram is only affected if definition changes
* Provide Modularity
  + Program can be broken down into manageable, well-defined logical modules
  + Supports top-down design and stepwise refinement approach to problem solving
* Aid in abstraction
  + Allow mental separation from particulars
  + Stubs allow programmers to defer definition of procedures/functions until main program is tested and debugged
* Procedure performs specific action
* Stored in database and can be invoked or called by any anonymous block
* Can take parameters
* Datatype specifier in parameter declaration must be unconstrained
* Has two parts
  + Specification
    - begins with keyword PROCEDURE, ends with procedure name or parameter list
  + Body
    - begins with keyword IS, ends with keyword END followed by optional procedure name

CREATE [OR REPLACE] PROCEDURE <procedurename> [(parameter1, … parameterN)]

IS

[local declarations]

BEGIN

executable statements;

[EXCEPTION

exception handlers]

END [<procedurename>];

parameter stands for

variable\_name [IN|OUT|IN OUT] datatype [{: = | DEFAULT} value]

* **When a procedure is created, Oracle automatically performs these steps**
  + **Compiles the procedure**
  + **Stores the compiled code**
  + **Stores the procedure in the database**
* **The PL/SQL compiler used to compile the code**
* **If an error occurs, the procedure is created but it is invalid**
* **Enforce Data Security**
  + **Can grant users access to a procedure that queries a table, but not grant access to the table itself**
* **Improve Database Performance**
  + **Less amount of information sent over network**
  + **Compilation not required to execute code**
  + **Procedure present in shared pool so retrieval from disk not required**
* **Memory Savings**
  + **Only one copy of procedure needs to be loaded in memory for execution by multiple users**
* **Increase in Development Productivity**
  + **Can avoid redundant coding and increase productivity, by writing a single procedure**
* **Integrity**
  + **Procedures need to be tested only once, to guarantee that it returns accurate results**

**Calling a Stored Procedure**

* **Can call a procedure in a PL/SQL statement**
  + **Example**

**branch\_sum(‘NYK’);**

* **Can call a procedure from SQL\*Plus**
  + **Example**

**SQL> EXECUTE branch\_sum(‘NYK’);**

**Examples of Procedures –**

1. **--Procedure without parameters**

create or replace procedure pr1

as

s emp.sal%type;

Begin

select sal into s

from emp

where ename = 'SMITH';

dbms\_output.put\_line(s);

end;

**Parameter Modes for Procedures and Functions**

* **Used to define the behavior of formal parameters**
* **Can be used with any subprogram**
* **Three parameter modes**
  + **IN (Default)**
  + **OUT**
  + **IN OUT**
* **IN**
  + allows values to be passed to the subprogram being called
  + inside the subprogram it acts like a constant
  + actual corresponding parameter can be a constant, literal, initialized variable or expression
  + can be initialized to default values

**2) --Supplying parameters to a procedure which are by default of IN type**

create or replace procedure pr2 **(En IN Emp.Empno%type,**

**Name Emp.ename%type,**

**S Emp.Sal%type)**

is

Begin

Insert into Emp (empno, ename, sal)

Values (En, Name,S);

dbms\_output.put\_line('One record inserted through procedure');

End;

/

En, Name and S are termed as “Formal Parameters”.

Exec Pr2 (123, 'Richards', 4000);

123, Richards and 4000 are termed as “Actual Parameters”.

3) **Giving default values to the parameters**

**Due to default value given the parameter becomes optional also.**

**But if any other value is given then it takes it.**

**a) Default Value for the last parameter (s).**

create or replace procedure pr3

(Eno emp.empno%type,

N emp.ename%type,

S emp.sal%type,

**dno emp.deptno%type DEFAULT 10**)

is

Begin

Insert into emp (empno,ename,sal,deptno)

values(Eno,N,S,dno);

dbms\_output.put\_line('Record inserted');

End;

**-- While executing**

**--exec pr3 (1,'o',800) -----> (No deptno parameter given!!!)**

**b) Default value for the other than last parameter:**

create or replace procedure **dept\_rec**

(p\_deptno in number,

**p\_dname in dept.dname%type default 'Accounts',** p\_loc dept.loc%type)

is

begin

# Insert into dept

values(p\_deptno, p\_dname, p\_loc);

dbms\_output.put\_line('Done');

end;

**To** **exceute such a parameter we have to use the named notation**

**exec dept\_rec(p\_loc => 'Pune', p\_deptno => 10);**

**Mutiple ways of passing the parameters --**

1. **Positional Method -- exec dept\_rec(20,'Purchase','Mumbai');**
2. **Named Method –** 
   1. **exec dept\_rec(p\_loc => 'Bangalore',p\_deptno => 30);**
   2. **exec dept\_rec(p\_deptno =>20, p\_dname => 'Inspection', p\_loc => 'Pune');**
   3. **exec dept\_rec(p\_dname =>'Stores', p\_loc => 'Mumbai', p\_deptno => 10);**
3. **Combination Method –**

*After positional there can be named parameters, but after named there cannot be positional parameters.*

**exec dept\_rec( 50, p\_loc =>'Mumbai', p\_dname =>'Marketing'); -- Correct**

**exec dept\_rec(p\_deptno =>60, 'Pune','Training'); -- Wrong**

**--Cannot give size to the parameters**

create or replace procedure pr4 (name **char(10),** marks **number**)

is

v char(10);

Begin

if marks >= 35 then

dbms\_output.put\_line('Passed');

else

dbms\_output.put\_line('Failed');

end if;

dbms\_output.put\_line(name);

End;

**OUT parameter**

* + **allows values to be returned to the caller of a subprogram**
  + **inside the subprogram it acts like an uninitialized variable**
  + **actual corresponding parameter must be a variable; it cannot be a constant or expression**
  + **its value cannot be assigned to another variable or reassigned to itself**

create or replace procedure pr5(Name IN varchar2, **Salary OUT number**)

Is

Begin

Select sal into **Salary**

from emp

where ename = Name;

End;

**--Steps for displaying the OUT parameter**

--1) Compiling the procedure.

--2) Declare the bind variable on SQL prompt as **variable payment number**

*-- Bind variables are of SQL\* plus environment which are used to hold the return*

*--value given by the procedure or function.*

--3)Now execute the proc

-- **exec pr5('SMITH', :payment)**

--4)To display the value of payment

-- **print payment**

*--5)Bind variables are session specific.Their existence is removed as the session --ends.*

**Using local variable for out parameter when procedure is called inside another block.**

create or replace procedure p1(n in emp.ename%type, **s out emp.sal%type**)

is

begin

select sal into s

from emp

where ename = n;

end;

declare

x emp.sal%type;

begin

p1('SMITH',**x);**

dbms\_output.put\_line(x);

end;

**Out Mode:**

The procedure will assign or return value to the Out Mode parameter.

The out mode parameter can be initially null.

While giving the "Actual Parameter" we have to pass a "Variable".

That Variable can be of Editor or can be of PLSQL block.

**When we are calling procedure independently** then we need to pass the "Variable" of "that" editor. It is known as "Bind Variable of that Editor".

**When we are calling procedure inside a PLSQL block** then we need to pass the "local variable" of that block or procedure.

**6) IN OUT parameter**

* + **allows initial values to be passed and returns updated values to the caller**
  + **inside the subprogram it acts like an initialized variable**
  + **actual corresponding parameter must be a variable; it cannot be a constant or expression**
  + **can be assigned a value and its value can be assigned to another variable**

a)

create or replace procedure pr6(**x IN OUT number**)

Is

Begin

x := (x \* x);

End;

**/\*pr6 procedure cannot be executed independently on sql prompt.**

**It has to be called inside a plsql block. It actually gives the square value to the variable of that plsql block.**

**In short IN OUT type of paramter makes a procedure similar to function, as the function also returns the value to the calling environment.\*/**

b)

declare

a number;

Begin

a := &a;

pr6(a);

**/\*When a is given as a parameter , it's status is of IN OUT. So IN means the user input value and OUT means the changes square figure due to the procedure pr6. After the procedure is called with a as parameter then a value gets changed. At this time a acts as a OUT parameter, since the procedure is giving the changed value to a.\*/**

dbms\_output.put\_line(a);

End;

**7) IN OUT example from with respect to database**

**a)**

create or replace procedure salrise(salary IN OUT number)

is

Begin

salary := salary + (salary \* 0.20);

End;

/\*Salrise procedure will increment the sal by 20% and give the value to the calling plsql block.\*/

**b)**

Declare

n emp.ename%type;

s emp.sal%type;

Begin

n := '&n';

select sal into s

from emp

where ename = n;

dbms\_output.put\_line('The old salary is ' || s);

/\*Now calling the procedure Salrise and giving s as a IN parameter\*/

**Salrise(s);**

/\*After giving the salary as a parameter the salary value gets incremented by 20% \*/

dbms\_output.put\_line('The changed salary is '|| s);

/\*updating the table\*/

Update emp

set sal = s

where ename = n;

dbms\_output.put\_line('Salary of ' || n || ' is updated in the table.');

Exception

When NO\_DATA\_FOUND then

dbms\_output.put\_line('No such name');

End;

--------------------------------------------------------------------------------------------------

**In Out Mode:**

We need to pass an "initialized variable" .

Procedure will "change" that value.

Finally we will get the "changed value".

Like Out Mode even in In Out mode we need to pass "variable" as actual parameter.

The only difference is that the variable will have some value before getting passed in the procedure.

**Declaring Sub procedures**

**Without Sub Procedures the code would be as shown below:**

Create or Replace Procedure Main\_Procedure

as

Begin

xxxxxxxx -- Different code

yyyyyyyyy -- Different code

zzzzzzzzzz -- Different code

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant private code**

**aaaaaaaaaaaaaaaaaaaaaaa – Redundant private code**

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant private code**

pppppppppppp -- Different code

qqqqqqqqqqqq -- Different code

rrrrrrrrrrrrrrrrr -- Different code

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant private code**

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant private code**

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant private code**

wwwwwwwwwwww -- Different code

eeeeeeeeeeeeeeeee -- Different code

ttttttttttttttttttttttttt -- Different code

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant private code**

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant private code**

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant private code**

uuuuuuuuuuuuuuuuu -- Different code

iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii -- Different code

hhhhhhhhhhhhhhhhh -- Different code

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant private code**

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant private code**

**aaaaaaaaaaaaaaaaaaaaaaa – Redundant private code**

End;

Instead of above design we can have the following design using sub procedures:

**Create or Replace Procedure Main\_Procedure**

**As**

**-- Local Variables Area**

Procedure **Redundant\_Private\_Code** **-- Sub Procedure**

As

Begin

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant code**

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant code**

**aaaaaaaaaaaaaaaaaaaaaaa -- Redundant code**

End;

**Begin**

xxxxxxxx -- Different code

yyyyyyyyy -- Different code

zzzzzzzzzz -- Different code

**Redundant\_Private\_Code** ; **-- Calling the Sub Procedure**

pppppppppppp -- Different code

qqqqqqqqqqqq -- Different code

rrrrrrrrrrrrrrrrr -- Different code

**Redundant\_Private\_Code** ; **-- Calling the Sub Procedure**

wwwwwwwwwwww -- Different code

eeeeeeeeeeeeeeeee -- Different code

ttttttttttttttttttttttttt -- Different code

**Redundant\_Private\_Code** ; **-- Calling the Sub Procedure**

uuuuuuuuuuuuuuuuu -- Different code

iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii -- Different code

hhhhhhhhhhhhhhhhh -- Different code

**Redundant\_Private\_Code** ; **-- Calling the Sub Procedure**

**End;**

/

**Actual Example:**

Create Table emp3

As

Select \* from emp;

Create table log\_table(username varchar2(30), log\_date date);

**Create or replace procedure delete\_emp3(p\_ename emp3.ename%type)**

**as**

**X number;**

**/\*Creating a sub procedure \*/**

**Procedure log\_emp**

**as**

**Begin**

**Insert into log\_table**

**Values(user, SYSDATE);**

**Commit;**

**End;**

**Begin**

**Delete from emp3**

**where ename = p\_ename;**

**log\_emp; -- Calling the Sub Procedure**

**End;**

select \* from log\_table; **-- Empty**

Exec delete\_emp3 ('SMITH')

select \* from log\_table;

**Exec log\_emp**  **-- Error**

**Exec delete\_emp3.log\_emp -- Error**

**Handled or Unhandled Exceptions for a procedure getting called in another.**

**The following examples depict the concept of propagation of errors within the procedures:**

create or replace procedure innerproc(x number, y number)

is

begin

dbms\_output.put\_line(x / y);

exception

when zero\_divide then

dbms\_output.put\_line('y cannot be negative');

end;

create or replace procedure outerproc

as

begin

dbms\_output.put\_line('AA');

/\*Calling the innerproc precedure.\*/

**innerproc(4,0);**

dbms\_output.put\_line('CC’);

exception

when **others** then

dbms\_output.put\_line('A Run Time Error has occurred.');

end;

If the innerproc procedure is not having any error handler then the error handler outer procedure gets executed.