Stored Procedure & Function.

Define:- Stored function

Stored function is similar to a procedure which is a block of statement perform specific task.

But only the difference is that function always return a value where procedure may or may not return a value.

syntax:-

```
create or replace function function_name([Parameter
return return_type is/as
declaration;
begin
// execution statements;
end; OR end function_name;
```

Calling a function

① with Anonymous block

```
declare

begin
//F calling;
// Printing output;

end;
```

② select function_name (Parameter_values) fromdua

Optional.

* Stored Procedure:-

create or replace Procedure Procedure_name
([Parameters]) is

// declaration;
begin
// execution;
end; or end Procedure_name;
/

Syntax for calling a procedure.

i) By using Anonymous block:

declare
==
begin
Procedure_name(Parameter_values);
↑ optional
// Printing statements;
end;
/

ii) exec Procedure_name (parameter_values);

or

execute Procedure_name (Parameter_values);

PLSQL stored Procedure is a PLSQL block which Performs one or more specific task.
The Procedure contain the header & body.

1) header :-
The header contains name of the procedure & Parameters & variables Pass to the procedure.

2) Body :-
The body contains declaration section, execution section and exception section similar to the genera PLSQL Block.

* Parameters

Types of Parameters
There are 3 types of Parameters use in PLSQL Procedure & function
① In
② out
③ Inout

① In :- In Parameter is used to pass a value two the sub program.
It is a default mode of parameter passing.
Example:

```
create or replace procedure P1 (a in int b in int) is
c int;
begin
c:=a+b;
dbms_output.Put_line('Addition=' ||c);
end P1;
/
```

Calling above Procedure.

```
exec P1(10,20);
```

2) out:-
An out Parameter return a value to the calling program.
example:-

step 1
```
create or replace procedure P1(a in int, b in int,
                                c out int) is
begin
c:=a+b;
dbms_output.put_line('Addition=' ||c);
end P1;
```

step 2
```
create or replace Procedure P2 is
 K int;
begin
P1(30,20,K);
end P2;
```

step 3
```
exec P2;
```

3) Inout:-
An Inout Parameter passes and initial value to the sub Program and returns an updated value to the caller.
example:-

```
create or replace Procedure P1(a in int, b inout int)

begin
b:=a+b;
dbms_output.put_line('Addition=' ||b);
end P1;
exec P1 (50,40);
```

Imp

* Types of Attribute :-

1) % type :-

% type attribute is use to declare a variable that having exactly some data type as that of any column persent in the database table.

Syntax:-

Variable_name table_name.column_name %type;

Ex.

s student.sname %type;

2) % rowtype :-

% row type attribute is use to hold data type of complete record (row or tuple) into a single variable user have declare a single variable that stores different values of different columns as well as user can get this values.

Syntax:-

Variable_name table_name %row type;

Ex.

s student %row type;

* select into statement :-

The result of SQL command giving a single row can be assign to a record variable, row type variable or list of scalar variables. This is done by writing the SQL command and adding into clause.

```
begin
select * into e from Employee, investment where
Employee.emp_id = Investment.emp_id and inv_name
='mutual fund';
dbms_output.put_line(e.emp_id||' '||e.emp_name||' '
||e.address);
end;
/
from calling procedure.
execute emp details;
```

Q.2 write a function which will return total investment amount of a perticular client.

→
```
create or replace function total amt (eno int) return int is
                          or eno Employee.emp_id %type
total int;
begin
Select sum(inv_amount) into total from Investment,
Employee where Employee.emp_id=investment.emp_id
and emp_id = eno;
return total;
end;
/

select total amt (12) from dual;
```

Q.2 client(client_no, client_name, address, birthdate)
Policy-info(Policy_no, discription, maturity_amt, Premium_amt, Pdate)

Relation bet^n client & Policy-info is many to many

→ RDB is as follows:-
client(client_no, client_name, address, birthdate)
Policy-info(Policy_no, discription, maturity-amt, Premium-amt, Pdate)
cP(client_no, Policy_no)

1) write a Procedure which will display all Policy details having Premium amt less than 5000.

→ create or replace Procedure details is
P. Policy-info %rowtype;
begin
select * into P from Policy-info where Premium amt <5000;
dbms-output.Put_line(P.Policy_no || ' ' || P.descript || ' ' || P.maturity_amt);
end;
/
for calling Procedure:
exec Pdetails;

2) write the function which will return total maturity_amt of Policies of a Particular client.

→ create or replace function total maturity (c no int) return int is
or
cno client.client_no%type
total int;
begin
select sum(maturity_amt) into total from Policy-no = cP. Policy_no and client.client.no = ~~client.n~~ cP.cli

and client.client-no = cno;
return total;
end;
/
For calling Procedure:-
select total maturity(1122) from dual;

Q.3 Game(Gname, no-of-players, coach-name);
Player(Pid, Pname, address, club-name);

relation between Game & Player is many to many.

→ RDB is as follows:
Game(Gname, no-of-players, coach-name)
Player(Pid, Pname, address, club_name)
GP(Gname, Pid)

write a Procedure which will display game details having no: of players more than 5.

→ create or replace gamedetails is
e name % rowtype
begin
select * into e from Game, Player where Game.Gname = GP.Gname and Player.Pid = GP.Pid and no-of-Players > 5;
dbms-output.put_line(e.Gname || ' ' || e.no-of-Players || ' ' || e.coach-name);
end;
For calling Procedure:
exec game details;