**Functions**

* Subprogram that returns a value
* Have a RETURN clause
* Stored in database and can be invoked or called by any anonymous block
* Generally take parameters
* Datatype specifier in parameter declaration must be unconstrained
* Has two parts
  + Specification
    - begins with keyword FUNCTION, ends with RETURN clause
  + Body
    - begins with keyword IS, ends with keyword END followed by optional function name

**Syntax –**

CREATE [OR REPLACE] FUNCTION <functionname> [(argument1, … argumentN)] RETURN datatype IS

[local declarations]

BEGIN

executable statements;

[EXCEPTION

exception handlers]

END [<functionname>];

argument stands for

variablename [IN|OUT|IN OUT] datatype [{:= | DEFAULT} value]

**Format –**

CREATE FUNCTION day\_fn(mday NUMBER)

RETURN CHAR IS

disp\_day CHAR(15);

BEGIN

executable statements;

RETURN disp\_day;

EXCEPTION

statements;

END day\_fn;

**Calling a Function**

* Can call a function as a PL/SQL statement
  + Example

chardays := day\_fn(3);

* Can call a function as part of an expression
  + Example

IF day\_fn(3) = ‘TUESDAY’ THEN

statements;

END IF;

**The RETURN Statement**

* Immediately completes execution of subprogram and returns control to caller
* In procedures
  + cannot contain an expression
  + returns control to the caller before normal end of procedure is reached
* In functions
  + must contain an expression, evaluated when it is executed

**Examples of Functions –**

**1)**

a)

create or replace function eo ( n number)

**RETURN char**

is

disp char(30);

Begin

If Mod(n,2) = 0 Then

disp := 'Number is even';

else

disp := 'Number is odd';

End If;

dbms\_output.put\_line(disp);

**RETURN disp;**

End;

Select eo(9) from dual;

OR

Declare

x number;

y char(30);

Begin

x := &x;

**/\*Calling the function eo, supplying the parameter x and giving the value to y\*/**

**y := eo(x);**

end;

**2) Database Oriented Example**

**Step 1: Creating the Function**

create or replace function Remarks ( x number)

RETURN char

is

disp char(30);

Begin

If x >= 70 Then

disp := 'Distinction';

Elsif x >= 60 Then

disp := 'First Class';

Elsif x >= 50 Then

disp := 'Second Class';

Elsif x >= 35 Then

disp := 'Passed';

Else

disp := 'Failed';

End If;

RETURN disp;

End;

**Step 2: Check the logic:**

select Remarks(56) from dual;

**Step 3: Using this function to insert values for a column of a table**

create table student

(name varchar2(10),

marks number,

**result char(30)**);

**Step 4:Directly calling the function inside Insert statement**

Insert into student

values('John', 90,  **Remarks(90)**); **-- Directly calling the function in the values clause**

**Step 5: Cross-check by querying the table**

Select \* from student;

**Calling a function in select statement**

create or replace function raise\_sal(s number)

return number is

begin

return s + 8000;

end;

Select ename,deptno,job,sal, **raise\_sal(sal)**

From emp;

**Scenario of DML statements in the Function Body:**

Create table emp\_fun

As

Select \* from emp;

create or replace function f1(s number)

return number

is

begin

**delete from emp\_fun;**

return 0;

end;

**Wrong usage of functions**

**select ename,sal,f1(sal) from emp;**

This function now can only assign value to a variable.

But not a good use case….

declare

y number;

begin

y := f1(3);

dbms\_output.put\_line('Value of y is ' || y);

end;

select \* from emp\_fun; -- No rows selected!!!!!

**Ideally a function body should NOT have DML statements, because it will restrict the usage of that function.**

**Function having select statement in it’s body can get called in the select statement of the same table or any different table.**

create or replace function highest

return number

is

x number;

begin

select max(sal) into x from **emp**;

return x;

end;

select ename, sal, highest, highest - sal "Diff" from **emp**;

*Applicable for displaying aggregate values alongwith non groupable columns.*

**3) Returning more than one value using OUT parameter**

a)

create or replace function getdetails

(p\_deptno number,

**p\_dname OUT varchar**)

**RETURN varchar**

as

vloc varchar2(30);

begin

select dname, loc into p\_dname, vloc

from dept

where deptno = p\_deptno;

**RETURN vloc**;

End;

b)

**-- First declare two bind variables location and deptname**

**--SQL>** variable deptname varchar2(100) **(size is imp)**

**--SQL>** variable location varchar2(100)

Begin

:location := getdetails(30, :deptname);

End;

**-- To see both the values**

print deptname location

**But that function cannot be called in Select Satement:**

**select getdetails(10, :deptname) from dual;**

**In the Exception block also return statement has to be given.**

|  |
| --- |
| create or replace function get\_answer(p\_num integer)  return integer  as  v\_den integer := 0;  begin  **return p\_num / v\_den;**  Exception  when others then  v\_den := 1;  end;  / |

select get\_answer(9) from dual; -- Error

|  |
| --- |
| create or replace function get\_answer(p\_num integer)  return integer  as  v\_den integer := 0;  begin  **return p\_num / 0;**  Exception  when others then  v\_den := 1;  **return p\_num / v\_den;**  end;  / |

select get\_answer(9) from dual; -- Works fine now!!!

**So, it means that inside the exception block also the return statement is mandatory for successful execution of the function.**