PL/SQL FUNCTIONS & PROCEDURES

- A Procedure / Function is a logically grouped set of SQL & PL/SQL statements that perform a specific task.
- A PL/Sql functions are typically used to compute a value. There are many built-in functions such as SysDate, Sum, Avg. & To_Date.
- Developers also create their own functions when writing applications.
- The code for a PL/Sql function must contain a Return statement.

Procedures & Functions are made up of

① A Declarative Part ② Executable Part

③ Exception handling Part.

7,8
gb 207

where do Stored Procedures & functions reside ?

Stored in the oracle database. Can be invoked or called by any application using that function.

Syntax of PL/SQL Functions

```
CREATE [OR REPLACE] Function function_name
( Parameter 1, Parameter 2) RETURN datatype
IS
DECLARE Variable, Constant etc;
BEGIN
    Executable Statements
    Return (Return value);
END;
```

- user defined
- Function header
- Can have 0 Parameter or n no, of Parameters
- IS: indicates ending of header function
- Execution Part.

Program 1 - Create and call a standalone function which returns the total no, of customers in the customer table.

```
Select * from customers ;

CREATE OR REPLACE Function TotalCustomers
    RETURN number IS
      Total number(2) := 0 ;
BEGIN
   SELECT count(*) into total From customers;
   RETURN total ;
END ;
```

Function Created

Calling a function to return/show total no, of customers

```
DECLARE
   c number(2);
BEGIN
   c := total_customers();
   dbms_output.put_line('Total no, of customers: ' || c);
END ;
```

Program 2 - Create & call a function to compute tax ~~of the~~ on the Salary of the employees.

```
Create a replace function ctax (Sal number)
   return number
   as
   Begin
      if Sal < 2000 then
         return Sal * .10 ;
```

else
    return Sal * .20;
end if;
end;

Function Created

How to use the Ctax() on the table;

~~Select * from customers~~;

Select ename, Sal, ctax(Sal) from customers;

PROCEDURES

- These Programs do not return a value directly. Mainly used to perform an action.

Parts of SubProgram Procedure

① Declarative Part ② Executable Part ③ Exception handling

Creating a Procedure (Syntax)

```
Create [OR Replace] Procedure Procedure_name
    [(Parameter_name [IN | OUT | INOUT] data type ...)]
(IS | AS)
BEGIN
    <Procedure body>
END
```

- IN represents that value that will be passed from outside (By default)
- Send values to Stored Procedure
- OUT represents the Parameter that will be used to return a value outside of the Procedure.
- used to get value from Stored Procedures
- Procedure body contains executable part.

Example (1) - The fallow example creates a simple procedure that displays the string 'Hello world' on the screen when exec

```
CREATE OR REPLACE PROCEDURE greetings
IS/AS
BEGIN
   dbms_output.put_line ('Hello world');
END
```

} Procedure Created

Executing a standalone Procedure

(a) using Execute keyword

(b) Callig name of the Procedure from PL/SQL block

(a) EXECUTE greetings ; o/p Hello world.

(b)
```
BEGIN
   greetings ;
END
```
o/p Hello world.

Deleting a standalone Procedure

Syntax

```
DROP PROCEDURE Procedure_name;
```

Example

```
DROP PROCEDURE greetings;
```

IN OUT Mode Example

An INOUT Parameter passes an initial value to a Subprogram & returns an updated value to the Caller.

IN & OUT Mode Example

Program(2) Find minimum of two values. Here Procedure takes two numbers using the IN Mode & returns their minimum using OUT Parameter

```
DECLARE
    a number ;
    b number ;
    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 ;
BEGIN
    a := 23 ;
    b := 45 ;
    findMin ( a, b, c ) ;
    dbms_output.put_line ('Minimum of (23,45) : ' || c ) ;
END ;
```

O/P - Minimum of (23,45) : 23

local

Program 3- How to Create PL/SQL Stored Procedure & call it ~~esp~~.

```
CREATE OR REPLACE PROCEDURE Rebellion IS
    var_name  varchar2(20) := 'Malinder';
    var_web   varchar2(20) := 'edu.com';
BEGIN
  DBMS_OUTPUT.PUT_line ('whats up friend ? I am' ||var_name || 'from'
                        || var_ web );
END Rebellion ;
```

O/P : Procedure Rebellion compiled

· Two ways to call a Procedure

① EXECUTE Rebellion ;
EXEC Rebellion ;

② BEGIN
Rebellion ;
END ;

Program 4 = How to Create PL/SQL Stored Procedures with Parameters

```
Create OR Replace Procedure emp_Sal (dep_id Number , Sal_raise num
IS ~~sal_raise~~ := ~~2000~~ ;
BEGIN
update employees SET Salary = Salary * Sal_raise where
                    department_Id = dep_Id ;
                    Column name
END
```

O/P : Procedure emp_Sal compiled

To call a Procedure

```
BEGIN
    emp_sal (10, 2);
END
```

Create table Employees (name varchar(20), department_id number(20), Salary Integer).

Practical-7 Packages. (10)

A Package is schema object that groups logically related PL/SQL types, variables, constants, subprograms, cursors & exceptions. A package is compiled & stored in database where many appl" can share its contents.

Packages

Package Specifications — Package Body

Create or replace package P1
Is
Procedure P1 (a number);
function F1 (b number)
Return number;
end;

Package Specification:
All objects are placed in specifications are called Public objects. Any subprogram not in package specification but coded in package is called private object.

```
CREATE PACKAGE Cust_sal AS
  PROCEDURE find_sal ( c_id customers.id%type);
  END Cust_sal;
/
```

Package created

Package Body:

```
CREATEOR REPLACE PACKAGE BODY Cust_sal AS
PROCEDURE find_sal (c_id customers.id%TYPE)
  IS
```

C_Sal Customers.salary%TYPE;
BEGIN
SELECT salary INTO C_sal
FROM Customers
WHERE id=C_id;
dbms_output.put_line('salary:'||C_sal);
END find_sal;
END cust_sal;
/
Package Body Created

Declare
num emp.empno%type:=num;
Begin
emp_sal.find_sal(num);
d.o.p('—'||num);
end;

Create or replace package body
P1
Is
Procedure P1(a number)
Is
Begin
dbms——(a);
end;
Function Ret—f;(b number)

Package specification:

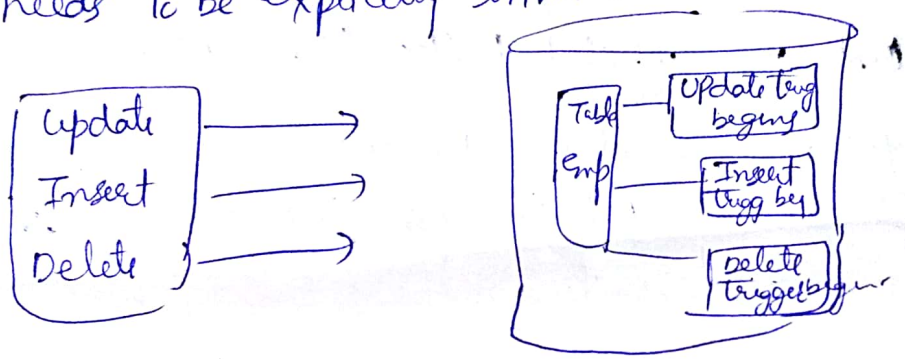
CREATE OR REPLACE PACKAGE C_package AS
-Adds a customer-
PROCEDURE addcustomer(C_id Customers.id%type,
C_name customers.name%type,
C_addr Customers.address%type,
C_sal Customers.Salary%type);
---Removes a customers-
PROCEDURE delcustomer(C_id Customers.id
%TYPE);
--List all customs
PROCEDURE listcustomer;
END C_package;

end;
end;

Begin
P1.P1(5);
end;
5

TRIGGERS

- A database trigger is a stored Procedure that is fired when an Insert, Update or delete statements is issued against the associated table.
- A Trigger can include SQL & PL/SQL statements to execute it as a unit.
- A Trigger is automatically executed without any action required by the user. A stored Procedure on other hand needs to be explicitly invoked.



Use of database Triggers

- To derive column values automatically
- To enforce complex integrity constraints

Database Trigger Vs Procedure

Trigger	Procedure
Trigger do not accept Parameters	Procedures can accept Parameters
A Trigger is automatically updated	A stored Procedure needs to be explicitly invoked.

Parts of a Trigger

1. Trigger event / Statement - is the SQL statement that causes a trigger to be fired

(2) Trigger constraint or Restrictions -
specifies a boolean expression that must be True for the trigger to fire. Trigger actions are not executed if the trigger restriction evaluates to False.

(3) Trigger action - is the Procedure that contains SQL statements & PL/SQL code to be executed when a triggering statement is issued.

TYPES OF TRIGGERS

1. Row Level Triggers - executes once for each row in a transaction. Commands of RLT are executed on all rows that are affected by the command.

Example - If a update statement updates multiple rows of a table, a row trigger is fired once for each row affected by the update statement

2. Statement level Triggers - SLT are triggered only once for each Transaction

For example - when an update command updates 15 rows, the commands contained in the trigger are executed only once.

SLT are the default types of trigger created via create Trigger command

3. Before & After Trigger - when trigger is defined, you can specify whether the trigger must occur befr or after the triggering event .ie Insert, update or Delete command

Name of Candidate: _____ Roll Number: _____

Date: _____ Batch Time: _____

SYNTAX For CREATING A TRIGGER


```
CREATE [OR REPLACE] TRIGGER trigger_name
Trig event { { BEFORE | AFTER }
           { [DELETE] | [OR] INSERT | [OR] UPDATE [OF column_name, ...] }
             ON table_name
[Referencing { OLD as old, NEW AS new }] } Triggering restrictions
[FOR Each Row [when condition ...]

DECLARE
    variable declaration ;
    Constant declaration ;
BEGIN
        PL/SQL SubProgram body ;
[ Exception
        exception PL/SQL block; ]
END
```

Triggering actions

Program 1 - To create a Trigger for the emp table, which makes the entry in EName column in upper case.

```
CREATE OR REPLACE Trigger upper_trigger
Before Insert OR Update of ename ON emp
For each row
BEGIN
    :new.ename := upper(:new.ename) ;
END ;
```

Program 2 - To create a trigger on the emp table which shows the old values & new value of ename after any updation on ename of emp table

```
Create or Replace Trigger Emp_Update
After update of EName on Emp for each Row
BEGIN
    DBMS_output.Put_line ('old Name : '||:old.Ename);
    ('New Name : '||:New.ename);
END;
```

Program : To create a trigger so that no operation can be performed on Emp table on Sunday 11/7/1981.

```
Create OR Replace Trigger Emp_Sunday
    Before Insert OR Update or Delete on Emp
BEGIN
  If RTRIM(upper(To_Char(SYSDATE,'Day.')))
        = 'SUNDAY' Then
  Raise_application_error(-20022,'No operation can be
                performed on Sunday');
  END if;
END;
```