**­­EXPERIMENT 8**

**AIM:** To develop procedures and function for various operations.

**FACILITIES REQUIRED­**

|  |  |  |
| --- | --- | --- |
| **Serial No.** | **Facilities required** | **Quantity** |
| 1 | System | 1 |
| 2 | Operating System | Windows |
| 3 | Front End |  |
| 4 | Backend | Oracle Apex |

**PL/SQL syntax**: A procedure is a block that can take parameters (sometimes referred to as arguments) and be invoked.

Procedures promote reusability and maintainability. Once validated, they can be used in number of applications. If the definition changes, only the procedure are affected, this greatly simplifies maintenance.

Modularized program development:

· Group logically related statements within blocks.

· Nest sub-blocks inside larger blocks to build powerful programs.

· Break down a complex problem into a set of manageable well defined logical modules and implement the modules with blocks.

**KEYWORDS AND THEIR PURPOSES**

**REPLACE:** It recreates the procedure if it already exists.

**PROCEDURE**: It is the name of the procedure to be created.

**ARGUMENT**: It is the name of the argument to the procedure. Parenthesis can be omitted if no arguments are present.

**IN**: Specifies that a value for the argument must be specified when calling the procedure ie.,      used to pass values to a sub-program. This is the default parameter.

**OUT**: Specifies that the procedure passes a value for this argument back to it’s calling environment after execution ie. used to return values to a caller of the sub-program.

**INOUT**: Specifies that a value for the argument must be specified when calling the procedure and that procedure passes a value for this argument back to it’s calling environment after execution.

**RETURN**: It is the data type of the function’s return value because every function must return a value, this clause is required.

**PROCEDURES**

**Syntax** : create or replace procedure (argument {in,out,inout} datatype ) {is,as}

variable declaration;

constant declaration;

begin

PL/SQL subprogram body;

exception exception PL/SQL block;

end;

**FUNCTIONS**

**Syntax**: create or replace function (argument in datatype,……) return datatype {is,as} variable declaration;

constant declaration;

begin

PL/SQL subprogram body;

exception

exception PL/SQL block;

end;

**Tables used:**

**SQL>** select \* from ititems\_171;

**ITEMID    ACTUALPRICE       ORDID        PRODID**

101            2000                         500                201

102            3000                        1600               202

103            4000                          600               202

**CREATE TABLE:**

create table ititems\_171(

    ITEMID number(10) primary key ,

    ACTUALPRICE number(10),

    ORDERID number(10),

    PRODID number(10)

 );

**INSERT VALUES**

INSERT INTO ITITEMS\_171 VALUES(101, 2000, 500, 201 );

INSERT INTO ITITEMS\_171 VALUES(102 ,3000 ,1600 ,202  );

INSERT INTO ITITEMS\_171 VALUES(103 ,4000 ,600 ,202  );

**PROGRAM FOR GENERAL PROCEDURE – SELECTED RECORD’S PRICE IS INCREMENTED BY 500 , EXECUTING THE PROCEDURE CREATED AND DISPLAYING THE UPDATED TABLE**

**SQL>**create procedure itemsum(identity number, total number) is price number;

 null\_price exception;

 begin

 select actualprice into price from ititems\_171 where itemid=identity;

 if price is null then

 raise null\_price;

 else

 update ititems\_171 set actualprice=actualprice+total where itemid=identity;

 end if;

 exception

 when null\_price then

 dbms\_output.put\_line('price is null');

 end;

/

**SQL>**BEGIN

              itemsum(101, 500);

         END;

**PROCEDURE FOR ‘IN’ PARAMETER – CREATION, EXECUTION**

**SQL>**CREATE OR REPLACE PROCEDURE yyy (a IN NUMBER) IS

price NUMBER;

BEGIN

BEGIN

SELECT actualprice INTO price

 FROM ititems\_171

WHERE itemid = a;

dbms\_output.put\_line('Actual price is ' || price);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

dbms\_output.put\_line('No item found for the given itemid');

price := NULL;

WHEN TOO\_MANY\_ROWS THEN

dbms\_output.put\_line('Error: More than one row returned');

price := NULL;

END;

IF price IS NULL THEN

dbms\_output.put\_line('Price is NULL');

END IF;

END;

/

**SQL>**BEGIN

    yyy(103);

       END

**PROCEDURE FOR ‘OUT’ PARAMETER – CREATION, EXECUTION**

**SQL>**CREATE OR REPLACE PROCEDURE zzz (c IN NUMBER, d OUT NUMBER) IS

          itemidentity NUMBER;

          BEGIN

          SELECT orderid INTO itemidentity

          FROM ititems\_171

          WHERE itemid = c;

          IF itemidentity < 1000 THEN

          d := 100;

          END IF;

          END;

          /

**SQL>**declare

  c  number;

  d number;

  begin

  zzz(101,d);

  dbms\_output.put\_line('The value of b is '|| d);

  end;

**PROCEDURE FOR ‘INOUT’ PARAMETER – CREATION, EXECUTION**

**SQL>**create procedure itit ( a in out number) is

   begin

   a:=a+1;

   end;

  /

**SQL>**declare

  a number:=7;

  begin

  itit(a);

  dbms\_output.put\_line('The updated value is '||a);

  end;

  /

**Tables used:**

**SQL>**SELECT\* FROM ITTRAIN

**TNO       TFARE**

1001        550

1002         600

**CREATE TABLE:**

create table ittrain(

    TNO number(10) primary key,

    TFARE number(10)

);

**INSERT TABLE:**

INSERT INTO ITTRAIN VALUES(1001, 550 );

INSERT INTO ITTRAIN VALUES(1002, 600 );

**PROGRAM FOR FUNCTION AND IT’S EXECUTION**

**SQL>**CREATE function trainfn (trainnumber number) return number is

  trainfunction ittrain.tfare % type;

  begin

  SELECT tfare into trainfunction from ITTRAINwhere tno=trainnumber;

  return(trainfunction);

  end;

  /

**SQL>**declare

  total number;

  begin

  total:=trainfn (1001);

  dbms\_output.put\_line('Train fare is Rs. '||total);

  end;

  /

**FACTORIAL OF A NUMBER USING FUNCTION — PROGRAM AND EXECUTION**

**SQL>**create function itfact (a number) return number is

 fact number:=1;

  b number;

  begin

  b:=a;

  while b>0

  loop

  fact:=fact\*b;

  b:=b-1;

  end loop;

  return(fact);

 end;

**/**

**SQL>**declare

  a number:=7;

  f number(10);

  begin

  f:=itfact(a);

  dbms\_output.put\_line('The factorial of the given number is '||f);

  end;

  /

**Q1: Write a procedure to calculate total for the all the students and pass regno, mark1, & mark2 as arguments.**

**Ans:SQL>**create table itstudent2\_171(regno number(3),name varchar(9),mark1 number(3),mark2 number(3));

**SQL>** insert into itstudent2\_171 values(110,'Arun',99,100)

 insert into itstudent2\_171 values(112,'Shiva',99,90)

**SQL>**select \* from itstudent2

**SQL>**alter table itstudent2\_171  add(total number(5));

**SQL>**select \* from itstudent2

**SQL>** create or replace procedure p1(sno number,mark1 number,mark2 number) is

tot number(5);

begin

tot:=mark1+mark2;

update itstudent2\_171 set total=tot where regno=sno;

end;

/

**SQL>** declare

 cursor c1 is select \* from itstudent2\_171;

 rec itstudent2\_171 % rowtype;

 begin

  open c1;

  loop

  fetch c1 into rec;

  exit when c1%notfound;

  p1(rec.regno,rec.mark1,rec.mark2);

  end loop;

  close c1;

  end;

  /

**SQL>**select \* from itstudent2\_171;

**Q2: Write a PL/SQL procedure called MULTI\_TABLE that takes two numbers as parameter and displays the multiplication of the first parameter till the second parameter.**

**Ans:SQL>**CREATE OR REPLACE PROCEDURE multi\_table (a NUMBER, b NUMBER) AS

     mul NUMBER;

BEGIN

     FOR i IN 1..b LOOP

         mul := a \* i;

DBMS\_OUTPUT.PUT\_LINE(a || '\*' || i || '=' || mul);

     END LOOP;

END;

/

**SQL>**declare

a number;

b number;

BEGIN

 multi\_table(:P1\_A, :P2\_B);

END;

/

**Q3: Consider the EMPLOYEE (EMPNO, SALARY, ENAME) Table. Write a procedure raise\_sal which increases the salary of an employee. It accepts an employee number and salary increase amount. It uses the employee number to find the current salary from the EMPLOYEE table and update the salary.**

**Ans:SQL>**CREATE OR REPLACE PROCEDURE raise\_sal(

 mempno emp\_disha\_171.empno%TYPE,

 msal\_percent NUMBER

) AS

BEGIN

UPDATE emp\_disha\_171

SET sal = sal + sal \* msal\_percent / 100

WHERE empno = mempno;

END;

/

**SQL>**SELECT \* from emp\_disha\_171

**SQL>**DECLARE

 CURSOR c1 IS SELECT \* FROM emp\_disha\_171;

 rec emp\_disha\_171%ROWTYPE;

BEGIN

OPEN c1;

 LOOP

 FETCH c1 INTO rec;

 EXIT WHEN c1%NOTFOUND;

 raise\_sal(rec.empno, 10);

 END LOOP;

 CLOSE c1;

 COMMIT;

END;

/

**SQL>**select\* from emp\_disha\_171

**Q4: Write a PL/SQL function CheckDiv that takes two numbers as arguments and returns the values 1 if the first argument passed to it is divisible by the second argument, else will return the value 0;**

**Ans:SQL>**create or replace function checkdiv (n1 number, n2 number) return number as res

number;

begin

if mod (n1, n2) = 0 then

res := 1;

else

res:= 0;

end if;

return res;

end;

/

**SQL>**DECLARE

     a NUMBER;

     b NUMBER;

BEGIN

     DBMS\_OUTPUT.PUT\_LINE('result=' || checkdiv(:P1\_a, :P2\_b));

END;

/

**Q5: Write a PL/SQL function called POW that takes two numbers as argument and return the value of the first number raised to the power of the second .**

**Ans:SQL>**create or replace function pow (n1 number, n2 number) return number as

res number;

begin

select power ( n1, n2) into res from dual; return res;

end;

**SQL>**declare

a number;

b number;

begin

dbms\_output.put\_line('power(n1,n2)='||pow(:P1\_a,:P1\_b));

end;

/

**Q6: Write a PL/SQL function ODDEVEN to return value TRUE if the number passed to it is EVEN else will return FALSE.**

**Ans:SQL>**create or replace function oddeven (n number) return boolean as

begin

if mod (n, 2) = 0 then return true;

else

return false;

end if;

end;

/

**SQL>**DECLARE

     a NUMBER;

     b BOOLEAN;

BEGIN

  a := :P1\_a;

         b := oddeven(a);

     IF b THEN

         DBMS\_OUTPUT.PUT\_LINE('The given number is Even');

     ELSE

DBMS\_OUTPUT.PUT\_LINE('The given number is Odd');

     END IF;

END;

/