Procedures And Functions

Procedures and Functions

- Procedures and functions:
  - Normally stored in the database within package specifications – a package is a sort of wrapper for a group of named blocks.
  - Can be stored as individual database objects.
  - Are parsed and compiled at the time they are stored.
  - Compiled objects execute faster than nonprocedural SQL scripts because nonprocedural scripts require extra time for compilation.

Procedures

- Procedures are named PL/SQL blocks.
- Created/owned by a particular schema
- Privilege to execute a specific procedure can be granted to or revoked from application users in order to control data access.
- Requires CREATE PROCEDURE (to create in your schema) or CREATE ANY PROCEDURE privilege (to create in other schemas).

CREATE PROCEDURE Syntax

```
CREATE [OR REPLACE] PROCEDURE <procedure_name> (<parameter1_name> <mode>
  <data type>, <parameter2_name> <mode> <data type>, ...) {AS|IS}
    <Variable declarations>
BEGIN
    Executable statements
[EXCEPTION
    Exception handlers]
END <optional procedure name>;
```

- Unique procedure name is required.
- OR REPLACE clause facilitates testing.
- Parameters are optional – enclosed in parentheses when used.
- AS or IS keyword is used – both work identically.
- Procedure variables are declared prior to the BEGIN keyword.
- DECLARE keyword is NOT used in named procedure.

Parameters

- Both procedures and functions can take parameters.
- Values passed as parameters to a procedure as arguments in a calling statement are termed *actual parameters*.
- The parameters in a procedure declaration are called *formal parameters*.
- The values stored in actual parameters are values passed to the formal parameters – the formal parameters are like placeholders to store the incoming values.
- When a procedure completes, the actual parameters are assigned the values of the formal parameters.
- A formal parameter can have one of three possible modes: (1) IN, (2), OUT, or (3) IN OUT.

Defining the IN, OUT, and IN OUT Parameter Modes

- **IN** – this parameter type is passed to a procedure as a read-only value that cannot be changed within the procedure – this is the default mode.
- **OUT** – this parameter type is write-only, and can only appear on the left side of an assignment statement in the procedure – it is assigned an initial value of NULL.
- **IN OUT** – this parameter type combines both IN and OUT; a parameter of this mode is passed to a procedure, and its value can be changed within the procedure.
- If a procedure raises an exception, the formal parameter values are not copied back to their corresponding actual parameters.

Procedure to find square of a given number

```
CREATE OR REPLACE PROCEDURE squareNum(x IN number ,square out number) IS
BEGIN
  square := x * x;
END;
/
```

To compile the procedure:

```
SQL> start F:\advdbms\2020\lab\program\procedure\proc_sqr.sql;
                              OR
SQL> @ F:\advdbms\2020\lab\program\procedure\proc_sqr.sql;

Procedure created.
```

Showing errors

```
 Warning: Procedure created with compilation errors.

SQL> select * from user_errors;

NAME         TYPE    SEQUENCE      LINE   POSITION TEXT ATTRIBUTE MESSAGE_NUMBER
--------- -------------
SQUARENUM  PROCEDURE       1         7         1
PLS-00103: Encountered the symbol ";"
ERROR               103

SQL> show errors;
Errors for PROCEDURE SQUARENUM:

LINE/COL ERROR
-------- -----------------------------------------------------------
7/1      PLS-00103: Encountered the symbol ";"
```

To execute Procedure

- Call procedure in a PL/SQL Program

```
DECLARE
sq number:=0;
x number:=&x;
BEGIN
  squareNum(x,sq);
  dbms_output.put_line(' Square is:'||sq);
END;
/
```

```
SQL> start F:\advdbms\2020\lab\program\procedure\call_sqr.sql;
Enter value for x: 4
old   3: x number:=&x;
new   3: x number:=4;
Square is:16

PL/SQL procedure successfully completed.
```

To execute Procedure

- **Use Exec/Execute**

```
SQL> var sqr number;
SQL> exec squarenum(8,:sqr);
PL/SQL procedure successfully completed.

SQL> print sqr;

       SQR
----------
        64
```

Procedure with No Parameters

```
SET SERVEROUTPUT ON
CREATE OR REPLACE PROCEDURE DisplaySalary IS
 temp_Salary NUMBER(10,2);
 temp_name emp.ename%type;
BEGIN
     SELECT Sal,ename INTO temp_Salary,temp_name FROM emp WHERE
  empno=102;
     DBMS_OUTPUT.PUT_LINE ('Salary of '||temp_name||' is
  '||temp_salary);
END;
/
```

Executing *DisplaySalary* Procedure

```
SQL> start F:\advdbms\2020\lab\program\procedure\proc_dissal.sql;

Procedure created.

SQL> execute displaysalary;
Salary of Ramesh is 35000

PL/SQL procedure successfully completed.
```

Passing IN and OUT Parameters

```
SET SERVEROUTPUT ON
CREATE OR REPLACE PROCEDURE displaysalary1(p_eno in number,p_sal
 out number) IS
BEGIN
    SELECT Sal INTO p_sal FROM emp WHERE empno=p_eno;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE ('Employee not found.');
END displaysalary1;
/
```

```
SQL> start F:\advdbms\2020\lab\program\procedure\proc_dissal1.sql;

Procedure created
```

Calling *DisplaySalary1*

```
DECLARE
v_sal emp.sal%type;
v_eno emp.empno%type:=&v_eno;
BEGIN
 displaysalary1(v_eno,v_sal);
 dbms_output.put_line('Actual salary of employee '||v_eno||' is
  '||v_sal);
END;
/
```

Executing

```
SQL> start F:\advdbms\2020\lab\program\procedure\call_dissal.sql;
Enter value for v_eno: 101
old   3: v_eno emp.empno%type:=&v_eno;
new   3: v_eno emp.empno%type:=101;
Actual salary of employee 101 is 30000


PL/SQL procedure successfully completed.


SQL> start F:\advdbms\2020\lab\program\procedure\call_dissal.sql;
Enter value for v_eno: 102
old   3: v_eno emp.empno%type:=&v_eno;
new   3: v_eno emp.empno%type:=102;
Actual salary of employee 102 is 35000
```

Executing using bind variables

```
SQL> var v_sal number;
SQL> execute displaysalary1(103,:v_sal);
PL/SQL procedure successfully completed.
SQL> print v_sal;

     V_SAL
----------
     55000
```

```
SQL> var v_sal number;
SQL> execute displaysalary1(100,:v_sal);
Employee not found.

PL/SQL procedure successfully completed.
```

Cursor in Procedure

```
SET SERVEROUTPUT ON
CREATE OR REPLACE PROCEDURE displaysalary2(p_dno in varchar2) IS
cursor cur_dis is select ename,sal from emp where deptno=p_dno;
BEGIN
for i in cur_dis
loop
dbms_output.put_line(i.ename||' earns '||i.sal);
end loop;
END displaysalary2;
/
```

Executing

```
SQL> start F:\advdbms\2020\lab\program\procedure\proc_dissal2.sql;

Procedure created.

SQL> execute displaysalary2('D1');
Sona earns 55000
Tina earns 25000

PL/SQL procedure successfully completed.
```

Dropping a Procedure

- The SQL statement to drop a procedure is the straight-forward DROP PROCEDURE <procedureName> command.
- This is a data definition language (DDL) command, and so an implicit commit executes prior to and immediately after the command.

```
SQL> DROP PROCEDURE DisplaySalary2;
Procedure dropped.
```

Create Function Syntax

- Like a procedure, a function can accept multiple parameters, and the data type of the return value must be declared in the header of the function.

```
CREATE [OR REPLACE] FUNCTION <function_name>
  (<parameter1_name> <mode> <data type>,
    <parameter2_name> <mode> <data type>, ...)
RETURN <function return value data type> {AS|IS}
    <Variable declarations>
BEGIN
    Executable Commands
    RETURN (return_value);
    . . .
[EXCEPTION
    Exception handlers]
END;
```

- The general syntax of the RETURN statement is:

```
RETURN <expression>;
```

Example1-To retrieve salary

```
SET SERVEROUTPUT ON
CREATE OR REPLACE function Display_Salary(v_eno in  number)
Return number IS
 temp_Salary NUMBER(10,2);
BEGIN
     SELECT Sal INTO temp_Salary FROM emp WHERE empno=v_eno;
     return temp_salary;
END;
/
```

Executing Functions

PL/SQL Block

```
DECLARE
v_sal emp.sal%type;
v_eno emp.empno%type:=&v_eno;
BEGIN
 v_sal:=display_salary(v_eno);
 dbms_output.put_line('Salary of '||v_eno||' is '||v_sal);
 END;
/
```

Executing Functions

PL/SQL Block

```
DECLARE
v_sal emp.sal%type;
v_eno emp.empno%type:=&v_eno;
BEGIN
 v_sal:=display_salary(v_eno);
 dbms_output.put_line('Salary of '||v_eno||' is '||v_sal);
 END;
/
```

```
SQL> start F:\advdbms\2020\lab\program\procedure\call_func.sql;
Enter value for v_eno: 103
old   3: v_eno emp.empno%type:=&v_eno;
new   3: v_eno emp.empno%type:=103;
Salary of 103 is 55000
```

Executing Functions

SELECT

```
SQL> select display_salary(103) from dual;

DISPLAY_SALARY(103)
-------------------
              55000
```

Executing Functions

EXECUTE/EXEC

```
SQL> var v_sal number;
SQL> exec :v_sal:=display_salary(103);

PL/SQL procedure successfully completed.

SQL> print v_sal;
     V_SAL
----------
     55000
```

Dropping a Function

- As with the DROP PROCEDURE statement, the DROP FUNCTION <functionName> is also straight-forward.
- As with DROP PROCEDURE, the DROP FUNCTION statement is a DDL command that causes execution of an implicit commit prior to and immediately after the command.

```
SQL> DROP FUNCTION FullName;
Function dropped.
```

Write a function to display the name of the employee drawing less salary in a given department

```
--function to display the name of employee earning less salary
SET SERVEROUTPUT ON
CREATE OR REPLACE function Display_ename(d_no in  varchar2)
Return varchar2 IS
 v_ename emp.ename%type;
BEGIN
    SELECT ename INTO v_ename FROM emp WHERE sal=(select min(sal) from emp
where deptno=d_no);
    return v_ename;
END;
/
```

```
SQL> start F:\advdbms\2020\lab\program\procedure\func_dispename.sql;

Function created.

SQL> select * from emp;

    EMPNO ENAME           SAL DEPT NO
---------- --------------- ---------- --
      101 Ravi             30000  D2
      102 Ramesh           35000  D2
      103 Sona             55000  D1
      104 Tina             25000  D1
      105 Bindu            35000  D3
      106 Bahabur          35000  D4

6 rows selected.
```

```
SQL> select display_ename('D2') from dual;

DISPLAY_ENAME('D2')
--------------------------------------------------------------------------------

Ravi

SQL> select display_ename('D1') from dual;

DISPLAY_ENAME('D1')
--------------------------------------------------------------------------------

Tina
```

Function Vs Procedure

Stored Procedure	**Function**
May or may not returns a value to the calling part of program.	Returns a value to the calling part of the program.
Uses IN, OUT, IN OUT parameter.	Uses only IN parameter.
Returns a value using “ OUT” parameter.	Returns a value using “RETURN”.
Does not specify the datatype of the value if it is going to return after a calling made to it.	Necessarily specifies the datatype of the value which it is going to return after a calling made to it.
Cannot be called from the function block of code.	Can be called from the procedure block of code.