Procedures and Functions

Mirunalini.P

SSNCE

March 9, 2021

Table of Contents

Session Objective

- To learn about procedures
- To learn about functions

PL/SQL

- PL/SQL is a block structured language
- The basic units (procedures, functions, and anonymous blocks) forms the PL/SQL program.
- PL/SQL program are logical blocks, which contain any number of nested subblocks.
- PL/SQL combines the data manipulating power of SQL with the data processing power of procedural languages.
- It uses the SQL data manipulation, cursor control, and transaction control commands, as well as the SQL functions and operators.
- Allows applications to be written in a PL/SQL procedure or a package and stored at Oracle server,

Parts of PL/SQL Block

PL/SQL has three parts:

- **A declarative part** : Items can be declared in declarative part.
- **An executable part** : Executable part is compulsory. Declared, items can be manipulated in the executable part.
- **An exception handling part** : Exceptions raised during execution can be dealt with in the exception handling part.

Block Structure

```
[ DECLARE
-- -        declarations
BEGIN
-- -        statements
[ EXCEPTION
handlers ]
  END ;
```

Example

```
declare
temp number(3);
begin
temp:=3;
dbms_ouput.put_line( 'Value of temp is =' || temp );
end;
```

dbms_output is an Oracle supplied PL/SQL package and put_line is one of the packaged procedures.

Must be enabled with SET SERVEROUTPUT ON command.

Stored Procedures

- SQL allows you to define procedures and functions and store them in the database server
- Executed by the database server.
- Oracle supports a slightly different version of Persistent Stored Modules (PSM) called PL/SQL

Stored Procedures

CREATE PROCEDURE <procedureName>
[(<paramList>)]
<localDeclarations>
<procedureBody>;

A parameter in the paramList is specified as:
<name> <mode> <type>
<mode> is one of {IN, OUT, INOUT}

eg: val1 IN int

Drop procedure by

DROP PROCEDURE $<procedureName>$

In PL/SQL, replace procedure by

CREATE OR REPLACE PROCEDURE <procedureName> ...

Example: Procedure in PSM

```
CREATE PROCEDURE testProcedure
BEGIN
INSERT INTO Student VALUES (5, ‘‘Joe’’);
END;
```

Oracle PL/SQL:

```
CREATE PROCEDURE testProcedure IS
BEGIN
INSERT INTO Student VALUES (5,"Joe");
END;
```

SHOW ERRORS to show the errors in your procedure.
call $< procedureName >$ [(< $paramList$ >)] – To call procedure

Example:

```
CREATE PROCEDURE testProcedure
(num IN int, name IN varchar) IS
num1 int; -- local variable
BEGIN
num1 := 10;
INSERT INTO Student VALUES (num, name);
END;
```

Control Structures: IF THEN ELSE

```
IF <condition> THEN

<statementList>

ELSIF <condition> THEN

<statementList>

ELSIF
...

ELSE <statementList>

END IF;
```

Loops

LOOP

<statementList>

END LOOP;

To exit from a loop use

EXIT;

Loops: Example

```
CREATE PROCEDURE testProcedure
(num IN int, name IN varchar) IS
num1 int;

BEGIN
num1 := 10;

LOOP
INSERT INTO Student VALUES (num, name);
num1 := num1 + 1;
IF (num1 > 15) THEN EXIT;
END IF;
END LOOP;

END;
```

FOR Loops

FOR i in [REVERSE] <lowerBound> .. <upperBound>

LOOP
<statementList>
END LOOP

Example:

```
FOR i in 1 .. 5 LOOP
INSERT INTO Student (sNumber) values (10 + i);
END LOOP;
```

WHILE LOOPS

```
WHILE <condition> LOOP
        <statementList>
END LOOP;
```

Functions

```
CREATE FUNCTION <functionName>
[(<paramList>)] RETURN type IS
<localDeclarations>
BEGIN <functionBody>;
....
END;
```

```
CREATE FUNCTION testFunction RETURN int IS
num1 int;
BEGIN
SELECT MAX (sNumber) INTO num1 FROM Student;
RETURN num1;
END;
```

SELECT * from Student where sNumber = testFunction ();

Cursors

- Oracle uses work areas to execute SQL statements and store processing information.
- A PL/SQL construct called a **cursor** name a work area and access its stored information.
- There are two kinds of cursors: **implicit and explicit**.
- PL/SQL declares a cursor implicitly for all SQL data manipulation statements, including queries that return only one row.
- For a queries that return multiple rows, you can explicitly declare a cursor to process the rows individually.

Implicit Cursors

- Oracle implicitly opens cursor to process each SQL statement not associated with an explicit cursor.
- PL/SQL lets you refer to the most recent implicit cursor as the SQL cursor, which always has these attributes:
  - **%FOUND** : This attribute yields TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it yields FALSE.
  - **%ISOPEN** : This attribute always yields FALSE because Oracle closes the SQL cursor automatically after executing its associated SQL statement.

Implicit Cursors

- **%NOTFOUND** : This attribute is the logical opposite of %FOUND.
- **%ROWCOUNT** : This attribute yields the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.
- These attributes were always used along with SQL:
  Ex: **SQL%FOUND, SQL%ROWCOUNT**
- **SQL** : This is the name of the **Oracle implicit cursor**

Implicit Cursors - Example

```
DECLARE
dept_no NUMBER(4) :=190;
BEGIN
DELETE FROM dept_temp WHERE department_id = dept_no;
IF SQL%FOUND THEN --> delete succeeded
INSERT INTO dept_temp VALUES (270, 'Personnel', 200, 1700);
END IF;
END;
```

Explict Cursors

- To execute a multirow query, Oracle opens an unnamed work area that stores processing information.
- For an explicit cursor give name for work area, **access the information, and process the rows individually**.
- **Declare a cursor** in the declarative part of any PL/SQL block, subprogram, or package.
- Use three commands to control a cursor: **OPEN, FETCH, and CLOSE**.
- Initialize the cursor with the **OPEN** statement, which identifies the resultset.
- Execute **FETCH** repeatedly until all rows have been retrieved.
- When the last row has been processed, Release the cursor with the **CLOSE** statement.

Explict Cursors - Example

```
CREATE OR REPLACE PROCEDURE copyProcedure IS
stID INT; name VARCHAR (10);
CURSOR myCursor IS SELECT * FROM STUDENT;
BEGIN
    OPEN myCursor;
LOOP
     FETCH myCursor INTO stID, name;
      EXIT WHEN myCURSOR%NOTFOUND;
      INSERT INTO newStudent VALUES (stID, name);
END LOOP;
CLOSE myCursor;
END;
```

PL/SQL Attributes

- PL/SQL variables and cursors have attributes which has the properties of referring the datatype and structure of an item without repeating its definition.
- A percent sign (%) serves as the attribute indicator. %TYPE
- The %TYPE attribute provides the datatype of a variable or database column.
- This is particularly useful when declaring variables that will hold database values.

```
credit REAL(7,2);
debit credit%TYPE;
```

- The %TYPE attribute is particularly useful when declaring variables that refer to database columns.
  my_title books.title% TYPE

PL/SQL Attributes - EXAMPLE

```
CREATE or Replace PROCEDURE addtuple2(
x T2.a%TYPE,
y T2.b%TYPE)
AS
BEGIN
INSERT INTO T2 VALUES (x, y);
END;
---calling procedure
Call addtuple2(10,'p');
```

%ROWTYPE

- The **%ROWTYPE** attribute provides a record type that represents a row in a table (or view).
- The record can store an entire row of data selected from the **table or fetched from a cursor.**

```
emp_rec emp%ROWTYPE (stores a row selected from the emp table)

CURSOR c1 IS SELECT deptno, dname, loc FROM dept;

dept_cur c1%ROWTYPE (stores a row fetched from cursor c1)
```

%ROWTYPE - EXAMPLE

```
DECLARE
CURSOR c1 IS SELECT last_name, salary FROM employees
 WHERE ROWNUM < 11;

my_ename employees.last_name%TYPE;
my_salary employees.salary%TYPE;
BEGIN
   OPEN c1;
LOOP
   FETCH c1 INTO my_ename, my_salary;
IF c1%FOUND THEN
  DBMS_OUTPUT.PUT_LINE('Name = ' || my_ename ||',
  salary = ' || my_salary);
ELSE
    EXIT;
END IF;
END LOOP;
```

Cursors - EXAMPLE

```
declare
        ename emp1.first_name%type;
        esal emp1.salary%type;
        num emp1.employee_id%type;
        inc number(6);
        begin
              num:=&eno;
              inc:=&increment;
              update emp1 set salary=salary+inc where
               employee_id=num;
```

Cursors - EXAMPLE

```
    if sql%found then
          select employee_id,first_name,salary into num,
          ename,esal from emp1 where employee_id=num;
          dbms_output.put_line('Number:'||num||
          ' Name'||ename||' Salary is.'||esal);
   else
          dbms_output.put_line('Record NOT FOUND !!!');
   end if;
end;
```

Cursors - EXAMPLE

```
declare
        emp emp1%rowtype;
        cursor c1 is select employee_id,first_name,salary
        from emp1 order by salary desc;
        counter number(2);
  begin
   for cur in c1 loop
    dbms_output.put_line(' Emp Number'||
    cur.employee_id||' Name '||
    cur.first_name||'Salary is '||cur.salary);
   end loop;
 end;
```

- Write a procedure to display all rows of the emp_table(e_no,e_fname, e_lname,start_date,end_date,salary,city,description)

Reference

 Fundamentals of Database systems 7^{th} Edition by Ramez Elmasri.