PL/SQL Programming – InterMediate

Hexaware TECHNOLOGIES

Course Objectives

- Procedures
- Functions

Session Objectives

- Procedures
- Functions

Procedures

Procedures

Understanding PL/SQL Procedures

A procedure is a PL/SQL subprogram used to perform an action. It can be stored in the database as a schema object for repeated execution.

```
CREATE [OR REPLACE]
PROCEDURE procedure_name[
(parameter1 mode1 datatype1, parameter2 mode2 datatype2...)]
IS | AS
[local declarations]
BEGIN
executable statements
[EXCEPTION
exception handlers]
END [name];
```

Procedures

- The CREATE clause enables the creation of Stand-alone procedures.
- The REPLACE option indicates,if the procedure already exists,it will dropped and replaced with the new version created by the statement.
- Procedure_name is the name of the stored procedure.
- Parameter is the name of the PL/SQL variable whose value is passed to or populated by the calling environment,based on the mode used.
- Mode is the Type of argument:
  - IN (default)
  - IN OUT
  - OUT
- Datatype is the datatype of the arugment-can be SQL or PL/SQL data type.Can be %TYPE,%ROWTYPE,any scalar or composite data type.The size of the datatype of the parameters cannot be restricted.

Procedures

- **Example of a simple PL/SQL procedure**

```
CREATE OR REPLACE
  PROCEDURE double ( original IN VARCHAR2,
                                  new_string OUT VARCHAR2 )
  AS
  BEGIN
     new_string := original || ' + ' || original;
  EXCEPTION
  WHEN VALUE_ERROR THEN
    DBMS_OUTPUT.PUT_LINE('Output buffer not long enough.');
  END double;
 /
```

Procedures

Example

```
CREATE OR REPLACE
  PROCEDURE proc1
    (p_emp_id IN employees.employee_id%TYPE)
  IS
    PROCEDURE proc2
    IS
    BEGIN
       …
    END;
 BEGIN
   …
   PROC2;
 END;
```

Procedure ‘PROC2’ declared inside another procedure (PROC1)

Procedures

Invoking Procedures

A procedure can be invoked in the following ways :

i. **EXECUTE [or EXEC] procedure_name;**

ii. **Invoking a procedure from an anonymous PL/SQL block**

```
DECLARE
  v_id NUMBER := 163;
BEGIN
  raise_salary(v_id);     --invoke procedure
  COMMIT;
...
END;
```

Procedures

ii. Invoking a procedure from another procedure

```
CREATE OR REPLACE PROCEDURE process_emps
IS
   CURSOR emp_cursor IS
    SELECT employee_id
    FROM   employees;
BEGIN
   FOR emp_rec IN emp_cursor
   LOOP
    raise_salary(emp_rec.employee_id);
   END LOOP;
   COMMIT;
END process_emps;
/
```

Procedures

Recompiling Procedures

Syntax

```
Alter procedure <procedure_name> compile;
```

Example

```
Alter procedure raise_salary compile;

Procedure altered.
```

Procedures Contd..

Removing Procedures

When a stored procedure is no longer required,it can be removed using the SQL DROP statement.

Syntax

```
DROP PROCEDURE procedure_name
```

Example

```
DROP PROCEDURE raise_salary;
```

Functions

Functions

Understanding PL/SQL Functions

A function is a PL/SQL subprogram used to perform an action.
It can be stored in the database as a schema object for repeated execution.

General Syntax for Creating Functions

```
CREATE [OR REPLACE]
FUNCTION function_name[
(parameter1 mode1 datatype1, parameter2 mode2 datatype2...)]
RETURN DATATYPE
IS | AS
[local declarations]
BEGIN
executable statements
[EXCEPTION
exception handlers]
END [name];
```

Functions Contd..

- The CREATE clause enables the creation of Stand-alone functions.
- The REPLACE option indicates, if the function already exists, it will dropped and replaced with the new version created by the statement.
- Function_name is the name of the stored function.
- Parameter is the name of the PL/SQL variable whose value is passed to the function.
- Type of the parameter. Only IN parameter should be declared.
- Datatype is the data type of the arugment.
- RETURN data type must be the data type of the output by the function.The data type must not include a size specifiction.There must be atleast one RETURN statement

Functions Contd..

- **Example of a simple PL/SQL function**

```
CREATE OR REPLACE FUNCTION employer_details_func2
 RETURN VARCHAR(20)
 IS
   emp_name VARCHAR(20)
 BEGIN
   SELECT first_name INTO emp_name FROM emp_tbl
   WHERE empID = '100';
   RETURN emp_name;
 END;
 /
```

Functions Contd..

Executing Functions

A function can be executed in the following ways :

1. Since a function returns a value we can assign it to a variable.

```
employee_name := employer_details_func;
```

If ‘employee_name’ is of datatype varchar we can store the name of the employee by assigning the return type of the function to it.

2. As a part of a SELECT statement

```
SELECT employer_details_func FROM dual;
```

3. In PL/SQL Statements like,

```
dbms_output.put_line(employer_details_func);
```

This line displays the value returned by the function.

Functions Contd..

Invoking User-defined functions from SQL Expressions

- PL/SQL user-defined functions can be called from SQL expressions.
- To be invoked from a SQL statement, a stored PL/SQL function must be declared either at schema level or in a package specification.

Advantages

- Permits calculations that are too complex ,awkward or unavailable in SQL.
- Increases data independence by processing complex data analysis in the Oracle server, rather than retrieving data into the application.
- Increases the efficiency of queries-Functions used in the WHERE clause of a query can filter data using criteria that must otherwise be evaluated by the application.

Functions Contd..

Example

The following user-defined function tax accepts a number as parameter and returns the tax by multiplying the parameter with 0.08.

```
CREATE OR REPLACE FUNCTION tax(p_value IN NUMBER)
 RETURN NUMBER IS
BEGIN
   RETURN (p_value * 0.08);
END tax;
/
SELECT employee_id, last_name, salary, tax(salary)
FROM   employees
WHERE  department_id = 100;
```

Function created.

EMPLOYEE_ID	LAST_NAME	SALARY	TAX(SALARY)
108	Greenberg	12000	960
109	Faviet	9000	720
110	Chen	8200	656
111	Sciarra	7700	616
112	Urman	7800	624
113	Popp	6900	552

6 rows selected.

Functions Contd..

- **Locations to call user-defined functions**

  A PL/SQL user-defined function can be called from any SQL expression wherever a built-in function can be called.

  For example, a PL/SQL function can appear in the following:

- Select list of the SELECT statement.
- Condition of the WHERE or HAVING clause.
- CONNECT BY, START WITH, ORDER BY, or GROUP BY clause.
- VALUES clause of the INSERT statement.
- SET clause of the UPDATE statement

Functions Contd..

```
SELECT employee_id,tax(salary)
FROM employees

WHERE
 tax(salary) > (SELECT MAX(tax(salary))
                FROM employees WHERE department_id=30)

ORDER BY tax(salary) DESC;
```

Functions Contd..

- **Restrictions on calling user-defined functions from SQL expressions**

To be callable from SQL expressions,PL/SQL functions must meet certain requirements :

- The function must be stored in the database.
- All of the function's parameters must use the IN mode and the positional notation.
- The datatypes of the function's parameters and the datatype of the RETURN clause of the function, must be recognized within the Oracle Server ((such as CHAR, DATE, or NUMBER), not PL/SQL data types (such as BOOLEAN, RECORD, or TABLE).
- The user must own or have an EXECUTE privilege on the function to call it from a SQL statement.
- A function called from SQL statement cannot contain DML statements.

Functions Contd..

- A function invoked from a DML statement cannot query the particular table being modified by that DML statement.
- A function invoked from a query or DML statement might not end the current transaction, create or rollback to a savepoint, or ALTER the system or session.
- A function cannot call another subprogram that breaks these restrictions.

NOTE :

Only Stored Functions are callable from SQL expressions. Stored Procedures cannot be called.

Functions Contd..

Removing Functions

When a stored function is no longer required, it can be removed using the SQL DROP statement.

Syntax

```
DROP FUNCTION function_name
```

Example

```
DROP FUNCTION tax;
```

Functions Contd..

- **Comparison Between a Procedure and a Function**

Procedure	Function
Normally used for executing business logic	Normally used for computations
Executed as a PL/SQL statement	Invoked as part of an expression
May or may not return a value or may return more than one value using the OUT parameter.	Must return a single value
No RETURN clause in the header and RETURN statement In the subprogram body	Must contain a RETURN clause in the header and a RETURN statement in the subprogram body
Cannot be called from SQL statements	Can be called from SQL statements

Best Practices

Best Practice	Benefits
Have single variable to hold the return value of a function.That variable is returned twice in the function.Once in the body as the last statement and once in the exception handler.	Ensures that the function returns only once.
Although OUT and INOUT parameters can be used in functions.It is not recommended.Use a procedure instead.	Ensures that the function returns only once.
Name the input and output parameters to the procedures and functions like p_i_salary(input) and p_o_tax (output).	Provides code clarity and aids in understanding

Best Practices

Best Practice	Benefits
Keep minimum constructs inside the package specification	-Reduces the risk of misuse. -Reduces the need for recompiling since changes to the package spec, require recompilation of dependent constructs

Thank You