More Package Concepts

Objectives

After completing this lesson, you should be able to do the following:

- Write packages that use the overloading feature
- Avoid errors with mutually referential subprograms
- Initialize variables with a one-time-only procedure
- Understand the purity of a function
- Understand persistent states

Overloading

- Allows you to use the same name for different subprograms inside a package
- Requires the formal parameters of the subprograms to differ in number, order, or datatype family
- Allows you to build more flexibility. A user or application is not restricted by the specific data type or number of formal parameters
- Restriction: Only local or packaged subprograms can be overloaded

Overloading: Example

```
CREATE OR REPLACE PACKAGE BODY over_pack
IS
  PROCEDURE add_dept
    (v_deptno  IN   dept.deptno%TYPE,
     v_name    IN   dept.dname%TYPE  DEFAULT 'unknown',
     v_loc     IN   dept.loc%TYPE    DEFAULT 'unknown')
  IS
  BEGIN
    INSERT INTO dept
    VALUES  (v_deptno,v_name,v_loc);
  END add_dept;
  PROCEDURE add_dept
    (v_name    IN  dept.dname%TYPE   DEFAULT 'unknown',
     v_loc     IN  dept.loc%TYPE     DEFAULT 'unknown')
  IS
  BEGIN
    INSERT INTO dept
    VALUES  (dept_deptno.NEXTVAL,v_name,v_loc);
  END add_dept;
END over_pack;
```

Forward Declarations

Identifiers must be declared before referencing them.

```
CREATE OR REPLACE PACKAGE BODY forward_pack
IS

   PROCEDURE award_bonus(. . .)
   IS
   BEGIN
    calc_rating(. . .);       --illegal reference

   END;

   PROCEDURE calc_rating(. . .)
   IS
   BEGIN

   END;

END forward_pack;
```

Forward Declarations

```
CREATE OR REPLACE PACKAGE BODY forward_pack
IS

 PROCEDURE calc_rating(. . .);    -- forward declaration

 PROCEDURE award_bonus(. . .)
 IS                               -- subprograms defined
 BEGIN                            -- in alphabetical order
  calc_rating(. . .);
 . . .
 END;

 PROCEDURE calc_rating(. . .)
 IS
 BEGIN
  . . .
 END;

END forward_pack;
```

Creating a One-Time-Only Procedure

```
CREATE OR REPLACE PACKAGE taxes
IS
   tax   NUMBER;
 ...  -- declare all public procedures/functions
END  taxes;

CREATE OR REPLACE PACKAGE BODY taxes
IS
 ... -- declare all private variables
 ... -- define public/private procedures/functions
BEGIN
  SELECT   rate_value
  INTO     tax
  FROM     tax_rates
  WHERE    rate_name = 'TAX';
END taxes;
```

Restrictions on Package Functions Used in SQL

- INSERT, UPDATE, or DELETE are not allowed.
- Only local functions can update package variables.
- Remote functions cannot read or write remote package variables.
- Functions that read or write package variables cannot use the parallel query option.
- Calls to subprograms that break the above restrictions are not allowed.

PL/SQL Compilation Checking

PRAGMA RESTRICT_REFERENCES:

- Checks that the restrictions of using functions in SQL statements are not violated
- Allows packaged functions to be used in SQL statements
- Verifies the purity of a function at compilation

PRAGMA RESTRICT_REFERENCES is no longer required for Oracle8*i*.

Purity Level of a Package Function

```
PRAGMA RESTRICT_REFERENCES (function_name,
                            WNDS
                           [,WNPS]
                           [,RNDS]
                           [,RNPS] );
```

PRAGMA RESTRICT_REFERENCES is no longer required for Oracle8*i*.

PRAGMA RESTRICT_REFERENCES

```
CREATE OR REPLACE PACKAGE taxes_pack
AS
  FUNCTION tax
    (v_value IN NUMBER)
     RETURN NUMBER;
  PRAGMA RESTRICT_REFERENCES (tax, WNDS, RNPS, WNPS);
END taxes_pack;
```

```
CREATE OR REPLACE PACKAGE BODY taxes_pack
 AS
   FUNCTION tax
     (v_value IN NUMBER)
      RETURN NUMBER
   IS
   BEGIN
     RETURN (v_value * 0.08);
   END tax;
 END taxes_pack;
```

Invoke a User-Defined Package Function from a SQL Statement

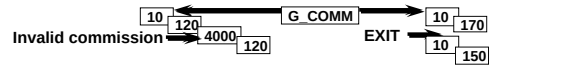
```
SQL> SELECT taxes_pack.tax(sal), sal,ename
  2  FROM   emp
  3  /

TAXES_PACK.TAX(SAL)        SAL ENAME
------------------- ---------- ----------
                400       5000 KING
                228       2850 BLAKE
                196       2450 CLARK
                238       2975 JONES
                100       1250 MARTIN
                128       1600 ALLEN
                120       1500 TURNER
                 76        950 JAMES
                100       1250 WARD
                240       3000 FORD
                 64        800 SMITH
                240       3000 SCOTT
                 88       1100 ADAMS
                104       1300 MILLER

14 rows selected.
```

Persistent State: Package Variables Example

```
CREATE OR REPLACE PACKAGE comm_package IS
  g_comm  NUMBER := 10;   --initialized to 10
  PROCEDURE reset_comm (v_comm   IN   NUMBER);
 END comm_package;
 CREATE OR REPLACE PACKAGE BODY comm_package IS
   FUNCTION  validate_comm (v_comm  IN  NUMBER)
RETURN BOOLEAN
   IS v_max_comm    NUMBER;
   BEGIN
    ... -- validates commission to be no more
    ... -- than maximum currently earned
   END validate_comm;
   PROCEDURE  reset_comm (v_comm   IN  NUMBER)
   IS BEGIN
     ... -- calls VALIDATE_COMM. If you try to reset
     ... -- the commission to be higher than the
     ... -- maximum, RAISE_APPLICATION_ERROR
   END reset_comm;
 END comm_package;
```

Persistent State: Package Variables

Time	Scott	Jones
09:00	EXECUTE comm_package.reset_comm(120)	
09:30		INSERT INTO emp (ename,comm) VALUES ('Madonna',2000);
09:35		EXECUTE comm_package.reset_comm(170)
10:00	EXECUTE comm_package.reset_comm(4000)	
11:00		ROLLBACK;
11:01		EXIT;
12:00		EXECUTE comm_package.reset_comm(150)



Persistent State: Package Cursor

Example

```
SQL> CREATE OR REPLACE PACKAGE pack_cur
  2  IS
  3     CURSOR c1 IS   SELECT   empno FROM    emp
  4                    ORDER BY empno DESC;
  5     PROCEDURE proc1_3rows;
  6     PROCEDURE proc4_6rows;
  7  END pack_cur;
  8  /
```

Persistent State: Package Cursor

```
SQL> CREATE OR REPLACE PACKAGE BODY pack_cur
  2  IS
  3    v_empno NUMBER;
  4    PROCEDURE proc1_3rows IS
  5    BEGIN
  6       OPEN c1;
  7      LOOP
  8        FETCH c1 INTO v_empno;
  9        DBMS_OUTPUT.PUT_LINE('Id :' ||(v_empno));
 10        EXIT WHEN c1%ROWCOUNT >= 3;
 11      END LOOP;
 12     END proc1_3rows;
 13    PROCEDURE proc4_6rows IS
 14    BEGIN
 15      LOOP
 16        FETCH c1 INTO v_empno;
 17        DBMS_OUTPUT.PUT_LINE('Id :' ||(v_empno));
 18        EXIT WHEN c1%ROWCOUNT >= 6;
 19      END LOOP;
 20      CLOSE c1;
 21    END proc4_6rows;
 22 END pack_cur;
 23 /
```

Persistent State: Package Cursor

```
SQL> SET SERVEROUTPUT ON
SQL> EXECUTE pack_cur.proc1_3rows
       Id : 7934
       Id : 7902
       Id : 7900
SQL> EXECUTE pack_cur.proc4_6rows
       Id : 7876
       Id : 7844
       Id : 7839
```

Persistent State: Package PL/SQL Tables and Records

```
CREATE OR REPLACE PACKAGE emp_package IS
  TYPE emp_table_type IS TABLE OF emp%ROWTYPE
     INDEX BY BINARY_INTEGER;
  PROCEDURE read_emp_table
   (emp_table  OUT  emp_table_type);
END emp_package;
```

```
CREATE OR REPLACE PACKAGE BODY emp_package IS
  PROCEDURE read_emp_table
   (emp_table  OUT  emp_table_type)
  IS
    i  BINARY_INTEGER := 0;
  BEGIN
    FOR emp_record IN (SELECT * FROM emp)
    LOOP
     emp_table(i) := emp_record;
     i:= i+1;
    END LOOP;
  END read_emp_table;
END emp_package;
```

Summary

In this lesson, you should have learned about the following features of packages:

- **Overloading**
- **Forward referencing**
- **One-time-only procedures**
- **Purity level of package functions**
- **Persistent state of packaged objects**