Database Programming with PL/SQL

Persistent State of Package Variables

ORACLE® ACADEMY

Objectives

This lesson covers the following objectives:

- Identify persistent states of package variables
- Control the persistent state of a package cursor

Purpose

Suppose you connect to the database and modify the value in a package variable, for example from 10 to 20. Later, you (or someone else) invoke the package again to read the value of the variable. What will you/they see: 10 or 20? It depends!

Real applications often invoke the same package many times. It is important to understand when the values in package variables are kept (persist) and when they are lost.

Package State

The collection of package variables and their current values define the package state. The package state is:

- Initialized when the package is first loaded
- Persistent (by default) for the life of the session
  - Stored in the session's private memory area
  - Unique to each session even if the second session is started by the same user
  - Subject to change when package subprograms are called or public variables are modified.

Other sessions each have their own package state, and do not see your changes.

Example of Package State

The following is a simple package that initializes a single global variable and contains a procedure to update it. SCOTT and JONES call the procedure to update the variable.

```
CREATE OR REPLACE PACKAGE pers_pkg IS
  g_var    NUMBER := 10;
  PROCEDURE upd_g_var (p_var IN NUMBER);
END pers_pkg;

CREATE OR REPLACE PACKAGE BODY pers_pkg IS
  PROCEDURE upd_g_var (p_var IN NUMBER) IS
    BEGIN
      g_var := p_var;
    END upd_g_var;
END pers_pkg;

GRANT EXECUTE ON pers_pkg TO SCOTT, JONES;
```

Example of Package State (cont.)

The following sequence of events occurs:

Time	Event	State for: Scott	Jones
9:00	`Scott> .. svar := pers_pkg.g_var;`	10	-

9:30	`Jones> .. jvar := pers_pks.g_var;`
	`Jones> .. pers_pkg.upd_g_var(20);`
	`Scott> .. svar := pers_pkg.g_var;`
		10	10
			20

9:35	`Scott> .. pers_pkg.upd_g_var(50);`
	`Jones> .. jvar := pers_pks.g_var;`
		50	20

10:00	Scott disconnects and reconnects in a new session		
10:05	`Scott> .. svar := pers_pkg.g_var;`	10	

Example of Package State (cont.)

Explanation of the events on the previous slide:

- At 9:00: Scott connects and reads the variable, seeing the initialized value 10.
- At 9:30: Jones connects and also reads the variable, also seeing the initialized value 10. At this point there are two separate and independent copies of the value, one in each session’s private memory area. Jones now updates his own session’s value to 20 using the procedure. Scott then re-reads the variable but does not see Jones’s change.

Example of Package State (cont.)

- At 9:35: Scott updates his own session's value to 50. Again, Jones cannot see the change.
- At 10:00: Scott disconnects and reconnects, creating a new session.
- At 10:05: Scott reads the variable and sees the initialized value 10**.**

These changes would not be visible in other sessions even if both sessions are connected under the same user name.

Persistent State of a Package Cursor

A cursor declared in the package specification is a type of global variable, and follows the same persistency rules as any other variable. A cursor's state is not defined by a single numeric or other value. A cursor's state consists of the following attributes:

- Whether the cursor is open or closed
- If open, how many rows have been fetched from the cursor (`%ROWCOUNT`) and whether the most recent fetch was successful (`%FOUND` or `%NOTFOUND`).

The next three slides show the definition of a cursor and its repeated use in a calling application.

Persistent State of a Package Cursor: Package Specification

The cursor declaration is declared globally within the package specification. Therefore, any or all of the package procedures can reference it.

```
CREATE OR REPLACE PACKAGE curs_pkg IS
  CURSOR emp_curs IS SELECT employee_id FROM employees
              ORDER BY employee_id;
    PROCEDURE open_curs;
  FUNCTION fetch_n_rows(n NUMBER := 1) RETURN BOOLEAN;
  PROCEDURE close_curs;
END curs_pkg;
```

Persistent State of a Package Cursor: Package Body

```
CREATE OR REPLACE PACKAGE BODY curs_pkg IS
  PROCEDURE open_curs IS
  BEGIN
    IF NOT emp_curs%ISOPEN THEN   OPEN emp_curs;  END IF;
  END open_curs;
  FUNCTION fetch_n_rows(n NUMBER := 1) RETURN BOOLEAN IS
    emp_id employees.employee_id%TYPE;
  BEGIN
    FOR count IN 1 .. n LOOP
      FETCH emp_curs INTO emp_id;
      EXIT WHEN emp_curs%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE('Id: ' ||(emp_id));
    END LOOP;
    RETURN emp_curs%FOUND;
  END fetch_n_rows;
  PROCEDURE close_curs IS BEGIN
    IF emp_curs%ISOPEN THEN  CLOSE emp_curs;  END IF;
  END close_curs;
END curs_pkg;
```

Invoking `CURS_PKG`

Step 1 opens the cursor. Step 2 (in a loop) fetches and displays the next three rows from the cursor until all rows have been fetched. Step 3 closes the cursor.

```
DECLARE
  v_more_rows_exist BOOLEAN := TRUE;
BEGIN
  curs_pkg.open_curs;                                  --1
  LOOP
    v_more_rows_exist := curs_pkg.fetch_n_rows(3);     --2
    DBMS_OUTPUT.PUT_LINE('-------');
    EXIT WHEN NOT v_more_rows_exist;
  END LOOP;
  curs_pkg.close_curs;                                 --3
END;
```

Invoking CURS_PKG (cont.)

- The first looped call to fetch_n_rows displays the first three rows. The second time round the loop, the next three rows are fetched and displayed. And so on.
- This technique is often used in applications that need to FETCH a large number of rows from a cursor, but can only display a few of them on the screen at a time.

```
DECLARE
  v_more_rows_exist BOOLEAN := TRUE;
BEGIN
  curs_pkg.open_curs;                                --1
  LOOP
    v_more_rows_exist := curs_pkg.fetch_n_rows(3);  --2
    DBMS_OUTPUT.PUT_LINE('-------');
    EXIT WHEN NOT v_more_rows_exist;
  END LOOP;
  curs_pkg.close_curs;                               --3
END;
```

Terminology

Key terms used in this lesson included:

- Package state

Summary

In this lesson, you should have learned how to:

- Identify persistent states of package variables
- Control the persistent state of a package cursor