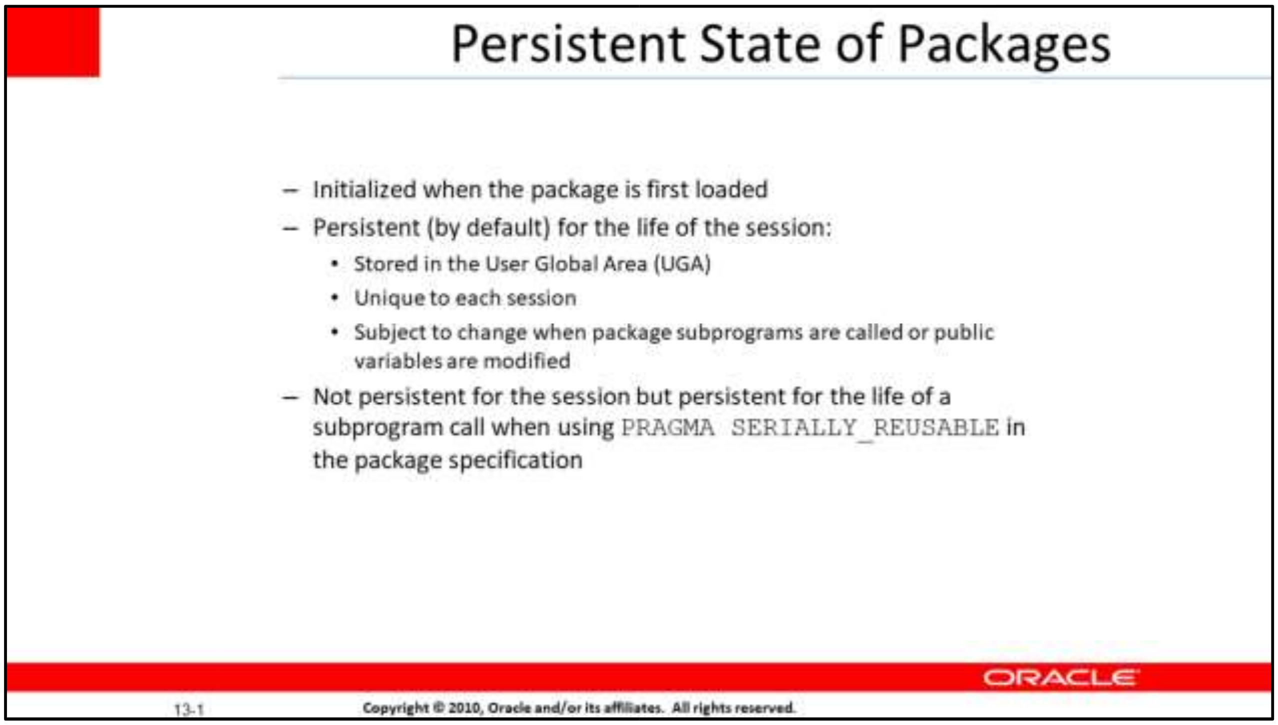


Persistent State of Packages

- Initialized when the package is first loaded
- Persistent (by default) for the life of the session:
  - Stored in the User Global Area (UGA)
  - Unique to each session
  - Subject to change when package subprograms are called or public variables are modified
- Not persistent for the session but persistent for the life of a subprogram call when using `PRAGMA SERIALLY_REUSABLE` in the package specification

13-1

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Persistent State of Packages

The collection of public and private package variables represents the package state for the user session That is, the package state is the set of values stored in all the package variables at a given point in time In general, the package state exists for the life of the user session

Package variables are initialized the first time a package is loaded into memory for a user session The package variables are, by default, unique to each session and hold their values until the user session is terminated In other words, the variables are stored in the User Global Area (UGA) memory allocated by the database for each user session The package state changes when a package subprogram is invoked and its logic modifies the variable state Public package state can be directly modified by operations appropriate to its type

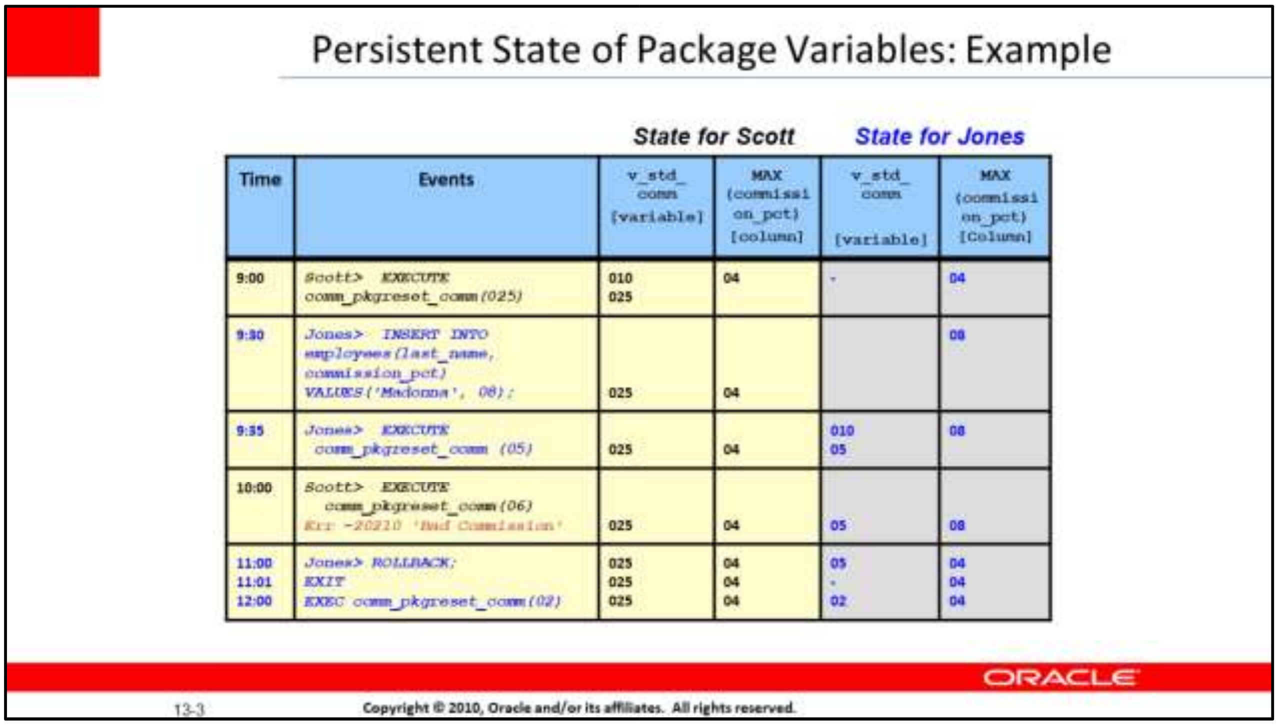
PRAGMA signifies that the statement is a compiler directive PRAGMAs are processed at compile time, not at run time They do not affect the meaning of a program; they simply convey information to the compiler If you add PRAGMA SERIALLY_RESUABLE to the package specification, then the database stores package variables in the System Global Area (SGA) shared across user sessions In this case, the package state is maintained for the life of a subprogram call or a single reference to a package construct The SERIALLY_REUSABLE directive is useful if you want to conserve memory and if the package state does not need to persist for each user session

Persistent State of Packages (continued)

This PRAGMA is appropriate for packages that declare large temporary work areas that are used once and not needed during subsequent database calls in the same session You can mark a bodiless package as serially reusable If a package has a spec and body, you must mark both You cannot mark only the body

The global memory for serially reusable packages is pooled in the System Global Area (SGA), not allocated to individual users in the User Global Area (UGA) That way, the package work area can be reused When the call to the server ends, the memory is returned to the pool Each time the package is reused, its public variables are initialized to their default values or to NULL

Note: Serially reusable packages cannot be accessed from database triggers or other PL/SQL subprograms that are called from SQL statements If you try, the Oracle server generates an error



Persistent State of Package Variables: Example

		State for Scott		State for Jones	
Time	Events	v_std_ comm [variable]	MAX (commission_pct) [column]	v_std_ comm [variable]	MAX (commission_pct) [Column]
9:00	Scott> EXECUTE comm_pkgreset_comm(025)	010 025	04	-	04
9:30	Jones> INSERT INTO employees(last_name, commission_pct) VALUES('Madonna', 08);	025	04		08
9:35	Jones> EXECUTE comm_pkgreset_comm (05)	025	04	010 05	08
10:00	Scott> EXECUTE comm_pkgreset_comm(06) Err -20210 'Bad Commission'	025	04	05	08
11:00 11:01 12:00	Jones> ROLLBACK; EXIT EXEC comm_pkgreset_comm(02)	025 025 025	04 04 04	05 - 02	04 04 04

13-3

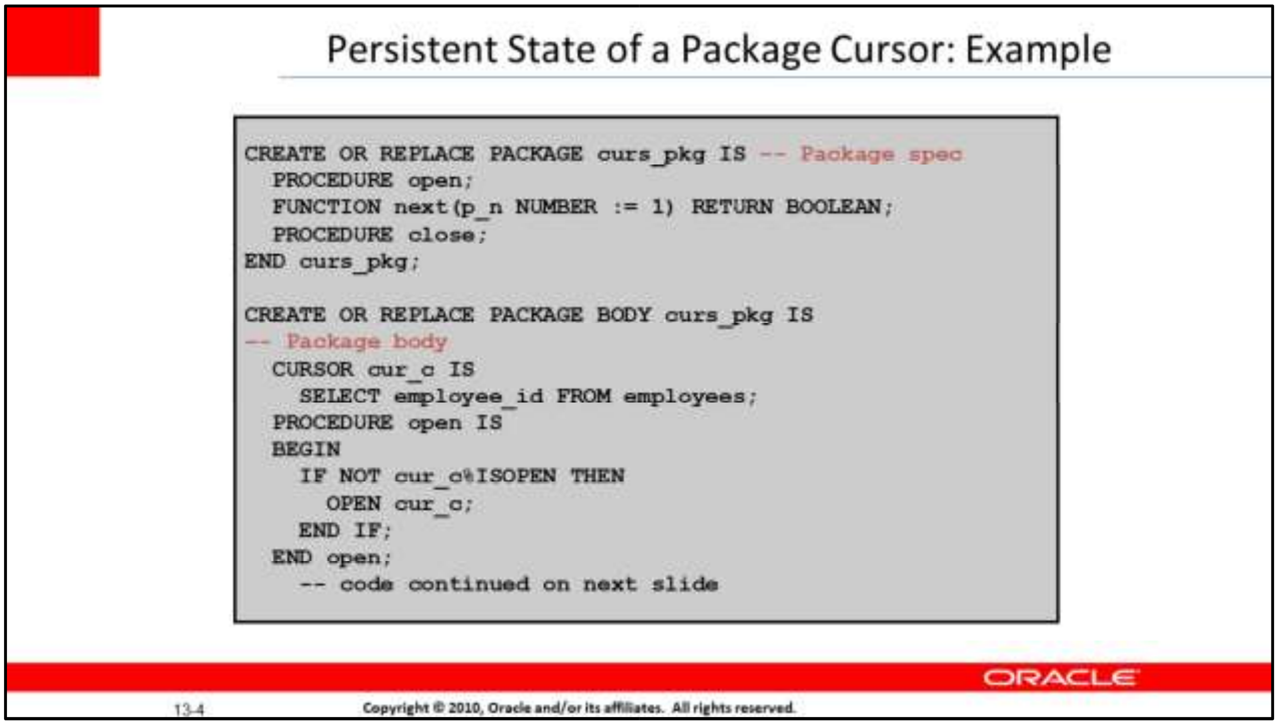
Persistent State of Package Variables: Example

The slide sequence is based on two different users, Scott and Jones, executing comm_pkg (covered in the lesson titled “Creating Packages”), in which the reset_comm procedure invokes the validate function to check the new commission The example shows how the persistent state of the v_std_comm package variable is maintained in each user session At 9:00: Scott calls reset_comm with a new commission value of 025, the package state for v_std_comm is initialized to 010 and then set to 025, which is validated because it is less than the database maximum value of 04 At 9:30: Jones inserts a new row into the EMPLOYEES table with a new maximum v_commission_pct value of 08 This is not committed, so it is visible to Jones only Scott’s state is unaffected

At 9:35: Jones calls reset_comm with a new commission value of 05 The state for Jones’s v_std_comm is first initialized to 010 and then set to the new value 05 that is valid for his session with the database maximum value of 08

At 10:00: Scott calls reset_comm with a new commission value of 06, which is greater than the maximum database commission visible to his session, that is, 04 (Jones did not commit the 08 value)

Between 11:00 and 12:00: Jones rolls back the transaction (INSERT statement) and exits the session Jones logs in at 11:45 and successfully executes the procedure, setting his state to 02



Persistent State of a Package Cursor: Example

```
CREATE OR REPLACE PACKAGE curs_pkg IS -- Package spec
  PROCEDURE open;
  FUNCTION next(p_n NUMBER := 1) RETURN BOOLEAN;
  PROCEDURE close;
END curs_pkg;

CREATE OR REPLACE PACKAGE BODY curs_pkg IS
-- Package body
  CURSOR cur_c IS
    SELECT employee_id FROM employees;
  PROCEDURE open IS
  BEGIN
    IF NOT cur_c%ISOPEN THEN
      OPEN cur_c;
    END IF;
  END open;
    -- code continued on next slide
```

13-4

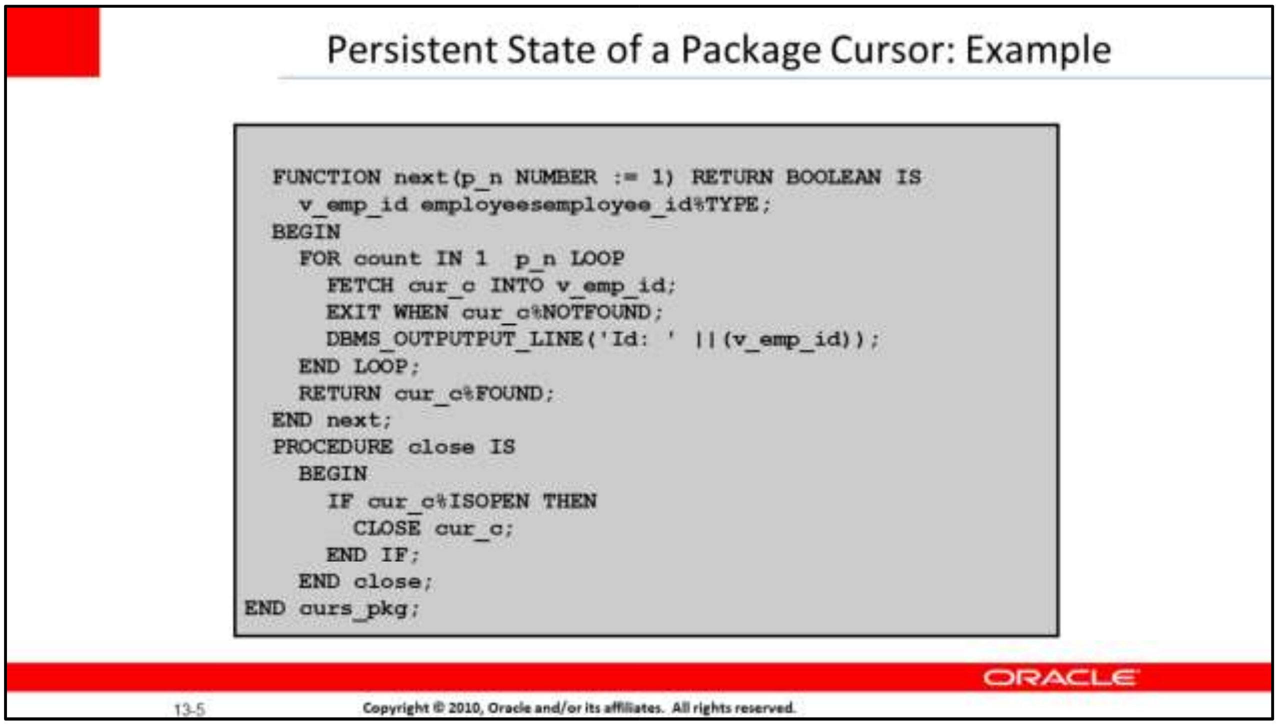
Persistent State of a Package Cursor: Example

The example in the slide shows the CURS_PKG package specification and body

The body declaration is continued in the next slide

To use this package, perform the following steps to process the rows:

1 Call the open procedure to open the cursor



Persistent State of a Package Cursor: Example

```
 FUNCTION next(p_n NUMBER := 1) RETURN BOOLEAN IS
   v_emp_id employeesemployee_id%TYPE;
 BEGIN
   FOR count IN 1  p_n LOOP
     FETCH cur_c INTO v_emp_id;
     EXIT WHEN cur_c%NOTFOUND;
     DBMS_OUTPUTPUT_LINE('Id: ' ||(v_emp_id));
   END LOOP;
   RETURN cur_c%FOUND;
 END next;
 PROCEDURE close IS
   BEGIN
     IF cur_c%ISOPEN THEN
       CLOSE cur_c;
     END IF;
   END close;
END curs_pkg;
```

13-5 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

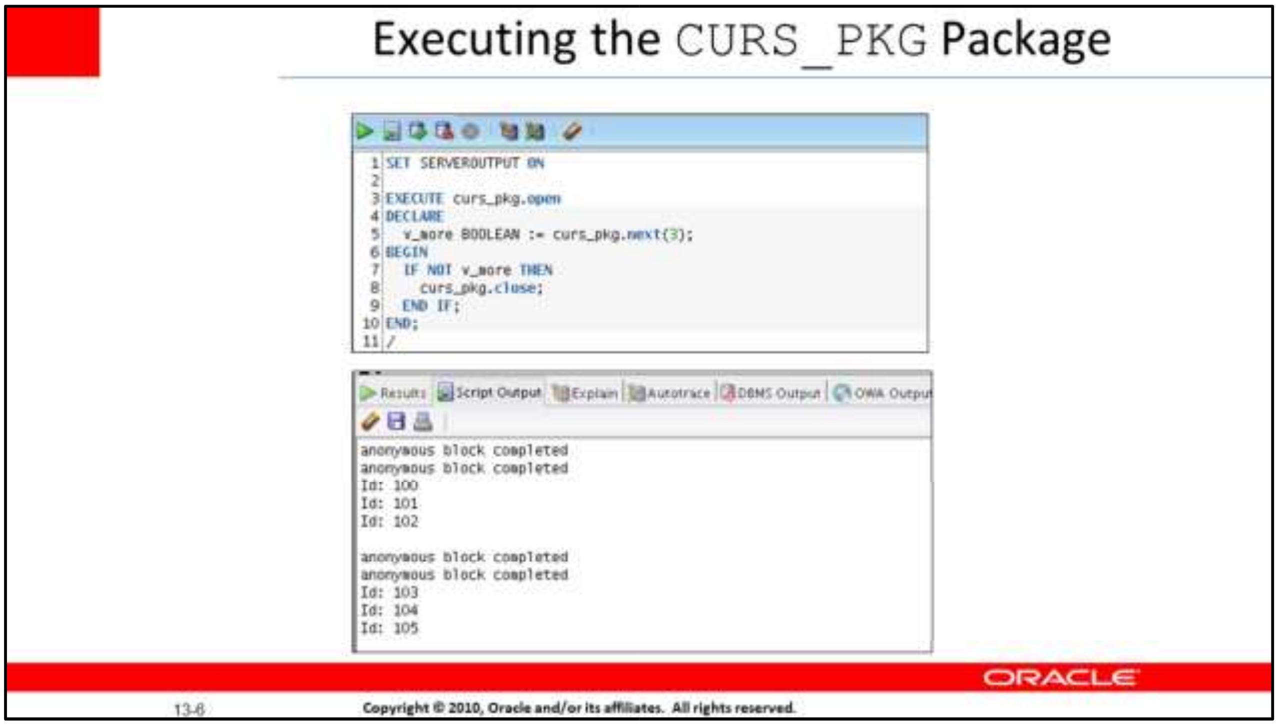
Persistent State of a Package Cursor: Example (continued)

2 Call the next procedure to fetch one or a specified number of rows If you request more rows than actually exist, the procedure successfully handles termination

It returns TRUE if more rows need to be processed; otherwise it returns FALSE

3 Call the close procedure to close the cursor, before or at the end of processing the rows

Note: The cursor declaration is private to the package Therefore, the cursor state can be influenced by invoking the package procedure and functions listed in the slide



Executing the CURS_PKG Package

```
SET SERVEROUTPUT ON

EXECUTE curs_pkg.open
DECLARE
  v_more BOOLEAN := curs_pkg.next(3);
BEGIN
  IF NOT v_more THEN
    curs_pkg.close;
  END IF;
END;
/
```

```
anonymous block completed
anonymous block completed
Id: 100
Id: 101
Id: 102

anonymous block completed
anonymous block completed
Id: 103
Id: 104
Id: 105
```

Executing CURS_PKG

Recall that the state of a package variable or cursor persists across transactions within a session However, the state does not persist across different sessions for the same user The database tables hold data that persists across sessions and users The call to curs_pkgopen opens the cursor, which remains open until the session is terminated, or the cursor is explicitly closed The anonymous block executes the next function in the Declaration section, initializing the BOOLEAN variable b_more to TRUE, as there are more than three rows in the EMPLOYEES table The block checks for the end of the result set and closes the cursor, if appropriate When the block executes, it displays the first three rows:

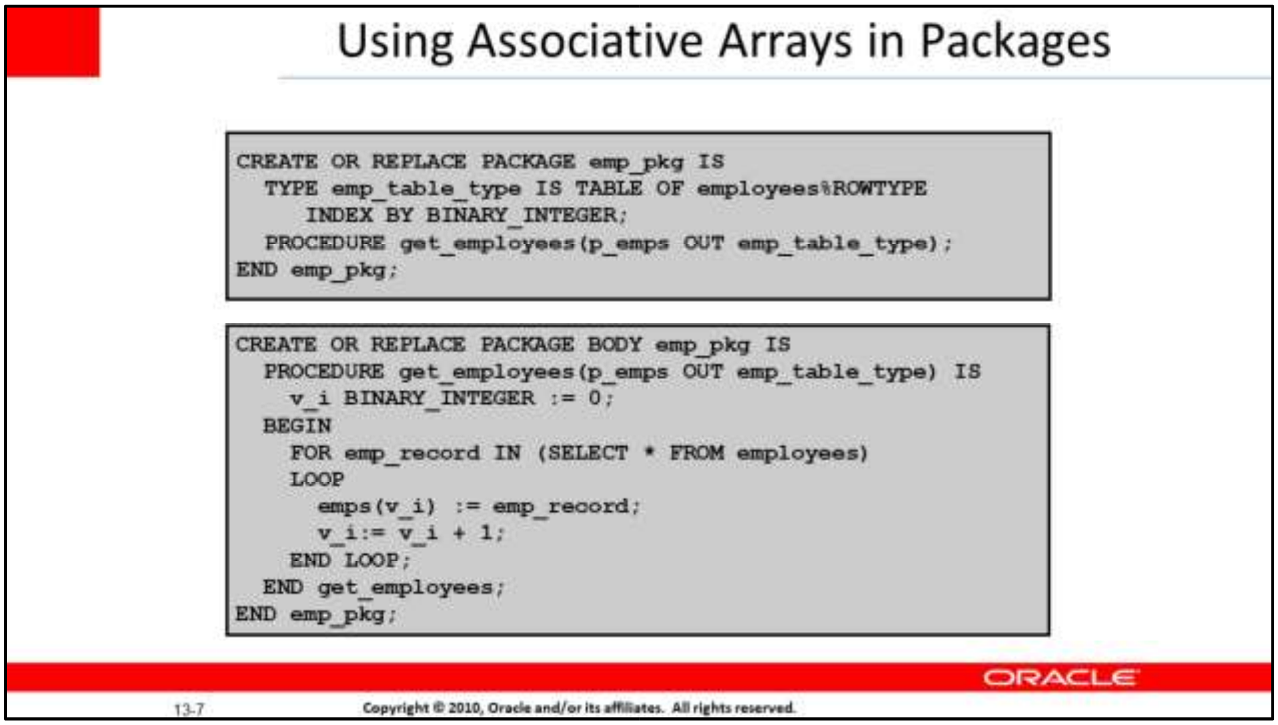
Id :100
Id :101
Id :102

If you click the Run Script (F5) icon again, the next three rows are displayed:

Id :103
Id :104
Id :105

To close the cursor, you can issue the following command to close the cursor in the package, or exit the session:

EXECUTE curs_pkgclose



Using Associative Arrays in Packages

```
CREATE OR REPLACE PACKAGE emp_pkg IS
  TYPE emp_table_type IS TABLE OF employees%ROWTYPE
     INDEX BY BINARY_INTEGER;
  PROCEDURE get_employees(p_emps OUT emp_table_type);
END emp_pkg;
```

```
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  PROCEDURE get_employees(p_emps OUT emp_table_type) IS
    v_i BINARY_INTEGER := 0;
  BEGIN
    FOR emp_record IN (SELECT * FROM employees)
    LOOP
      emps(v_i) := emp_record;
      v_i:= v_i + 1;
    END LOOP;
  END get_employees;
END emp_pkg;
```

ORACLE

13-7 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Associative Arrays in Packages

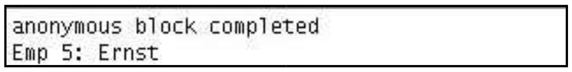
Associative arrays used to be known as index by tables

The emp_pkg package contains a get_employees procedure that reads rows from the EMPLOYEES table and returns the rows using the OUT parameter, which is an associative array (PL/SQL table of records) The key points include the following:

- employee_table_type is declared as a public type
- employee_table_type is used for a formal output parameter in the procedure, and the employees variable in the calling block (shown below)

In SQL Developer, you can invoke the get_employees procedure in an anonymous PL/SQL block by using the v_employees variable, as shown in the following example and output:

```
SET SERVEROUTPUT ON
  DECLARE v_employees
   emp_pkgemp_table_type;
  BEGIN
    emp_pkgget_employees(v_employees);
    DBMS_OUTPUTPUT_LINE('Emp 5:
```



```
anonymous block completed
Emp 5: Ernst
```

```
'||v_employees(4)last_name);
END;
```