13

More Package Concepts

Objectives

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

- Write packages that use the overloading feature
- Describe errors with mutually referential subprograms
- Initialize variables with a one-time-only procedure
- Identify persistent states

Overloading

- Enables you to use the same name for different subprograms inside a PL/SQL block, a subprogram, or a package
- Requires the formal parameters of the subprograms to differ in number, order, or data type family
- Enables you to build more flexibility because a user or application is not restricted by the specific data type or number of formal parameters

Note: Only local or packaged subprograms can be overloaded. You cannot overload stand-alone subprograms.

```
SET SERVEROUTPUT ON
CREATE OR REPLACE PACKAGE overload IS
   procedure p (x number);
   procedure p (n number);
END;
/
CREATE OR REPLACE PACKAGE BODY overload IS
   PROCEDURE p  (x number)  IS
     BEGIN
        DBMS_OUTPUT.PUT_LINE ('Tham so hinh thuc x :'||x);
   END;
   PROCEDURE p  (n number)  IS
      BEGIN
         DBMS_OUTPUT.PUT_LINE ('Tham so hinh thuc n :'||n);
      END;
END overload;
/
EXECUTE overload.p (x=>4);
EXECUTE overload.p (n=>3);
```

Overloading: Example

over_pack.sql

```
CREATE OR REPLACE PACKAGE over_pack
IS
PROCEDURE  add_dept
  (p_deptno IN departments.department_id%TYPE,
   p_name IN departments.department_name%TYPE
                                              DEFAULT 'unknown',
   p_loc IN departments.location_id%TYPE DEFAULT 0);
PROCEDURE  add_dept
  (p_name IN departments.department_name%TYPE
                                              DEFAULT 'unknown',
   p_loc IN departments.location_id%TYPE DEFAULT 0);
END over_pack;
/
```

```
Package created
```

Overloading: Example

over_pack_body.sql

```
CREATE OR REPLACE PACKAGE BODY over_pack IS
PROCEDURE add_dept
    (p_deptno IN departments.department_id%TYPE,
     p_name IN departments.department_name%TYPE DEFAULT 'unknown',
     p_loc IN departments.location_id%TYPE DEFAULT 0)
IS
BEGIN
  INSERT INTO departments (department_id, department_name, location_id)
  VALUES (p_deptno, p_name, p_loc);
END add_dept;
PROCEDURE add_dept
    (p_name IN departments.department_name%TYPE DEFAULT 'unknown',
     p_loc IN departments.location_id%TYPE DEFAULT 0)
IS
BEGIN
  INSERT INTO departments (department_id, department_name, location_id)
  VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_dept;
END over_pack;
/
```

Overloading: Example

- Most built-in functions are overloaded.
- For example, see the TO_CHAR function of the STANDARD package.

```
FUNCTION TO_CHAR (p1 DATE) RETURN VARCHAR2;
FUNCTION TO_CHAR (p2 NUMBER) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 DATE, P2 VARCHAR2) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 NUMBER, P2 VARCHAR2) RETURN VARCHAR2;
```

- If you redeclare a built-in subprogram in a PL/SQL program, your local declaration overrides the global declaration.

Using Forward Declarations

You must declare identifiers 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;
/
```

Using 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;
/
```

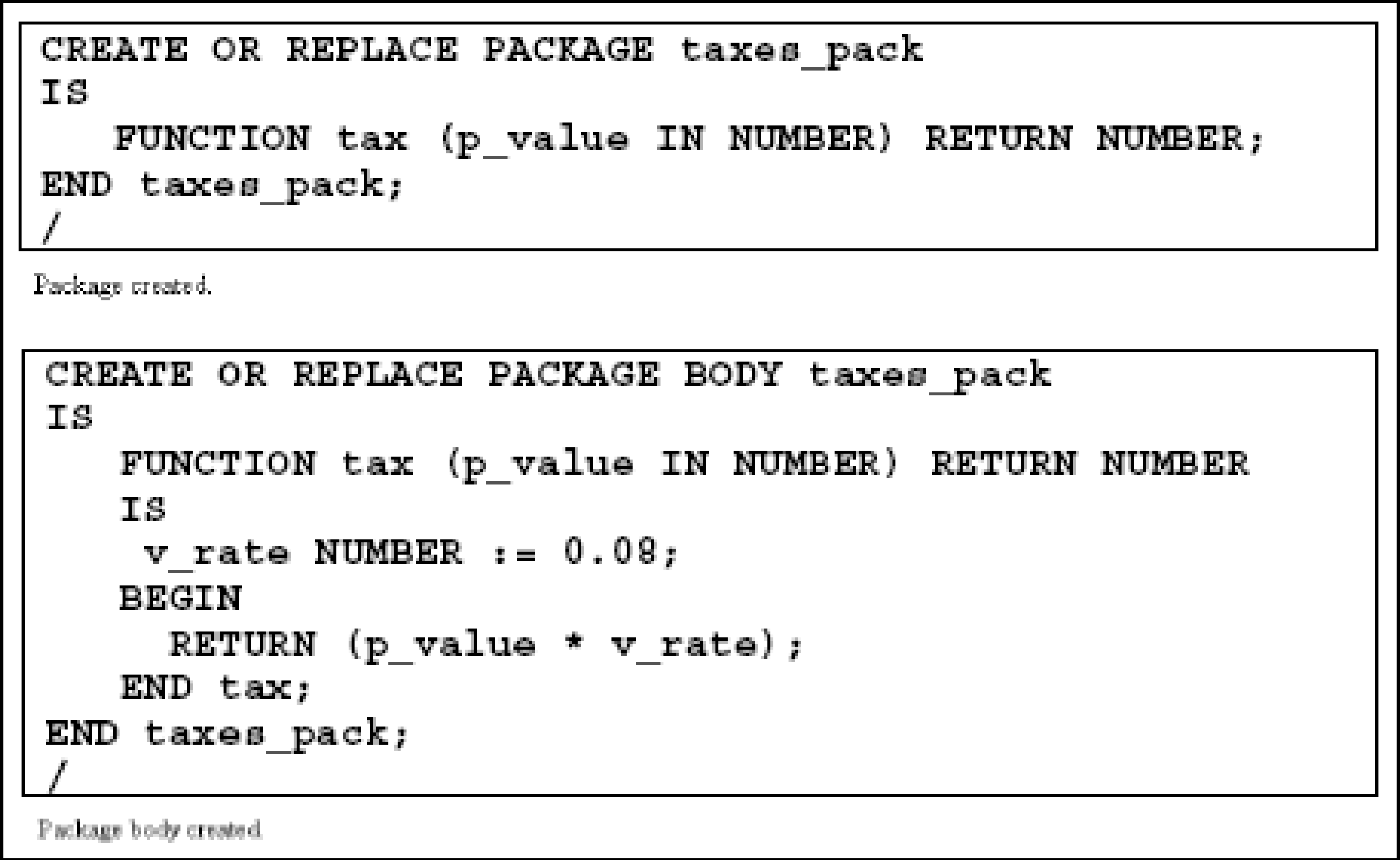
Restrictions on Package Functions Used in SQL

A function called from:

- A query or DML statement can not end the current transaction, create or roll back to a savepoint, or ALTER the system or session.
- A query statement or a parallelized DML statement can not execute a DML statement or modify the database.
- A DML statement can not read or modify the particular table being modified by that DML statement.

Note: Calls to subprograms that break the above restrictions are not allowed.

User Defined Package: taxes_pack



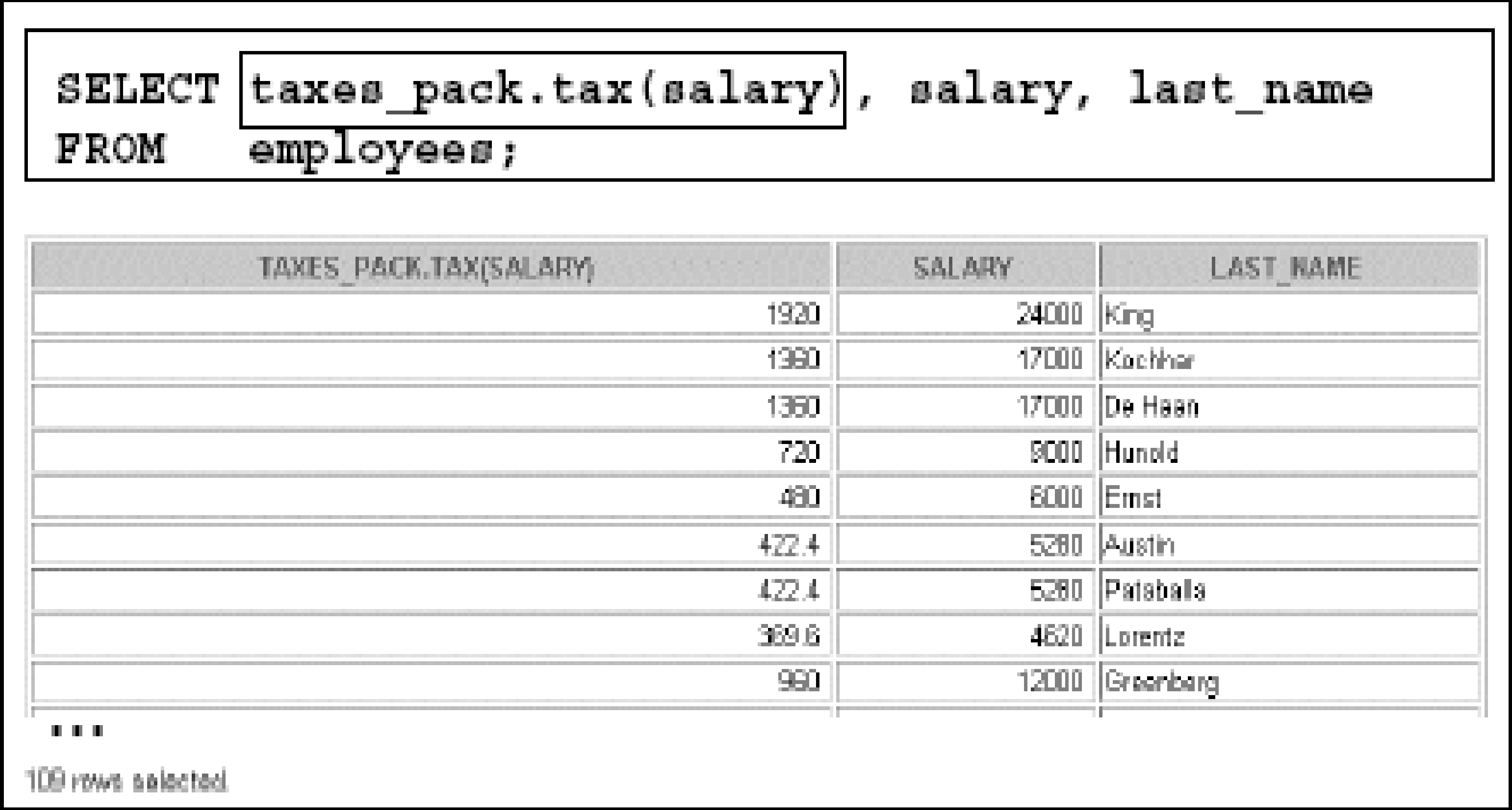
```
CREATE OR REPLACE PACKAGE taxes_pack
IS
   FUNCTION tax (p_value IN NUMBER) RETURN NUMBER;
END taxes_pack;
/
```

Package created.

```
CREATE OR REPLACE PACKAGE BODY taxes_pack
IS
   FUNCTION tax (p_value IN NUMBER) RETURN NUMBER
   IS
    v_rate NUMBER := 0.08;
   BEGIN
     RETURN (p_value * v_rate);
   END tax;
END taxes_pack;
/
```

Package body created.

Invoking a User-Defined Package Function from a SQL Statement



```
SELECT taxes_pack.tax(salary), salary, last_name
FROM   employees;
```

TAXES_PACK.TAX(SALARY)	SALARY	LAST_NAME
1920	24000	King
1360	17000	Kochhar
1360	17000	De Haan
720	9000	Hunold
480	6000	Ernst
422.4	5280	Austin
422.4	5280	Pataballa
369.6	4620	Lorentz
960	12000	Greenberg

...

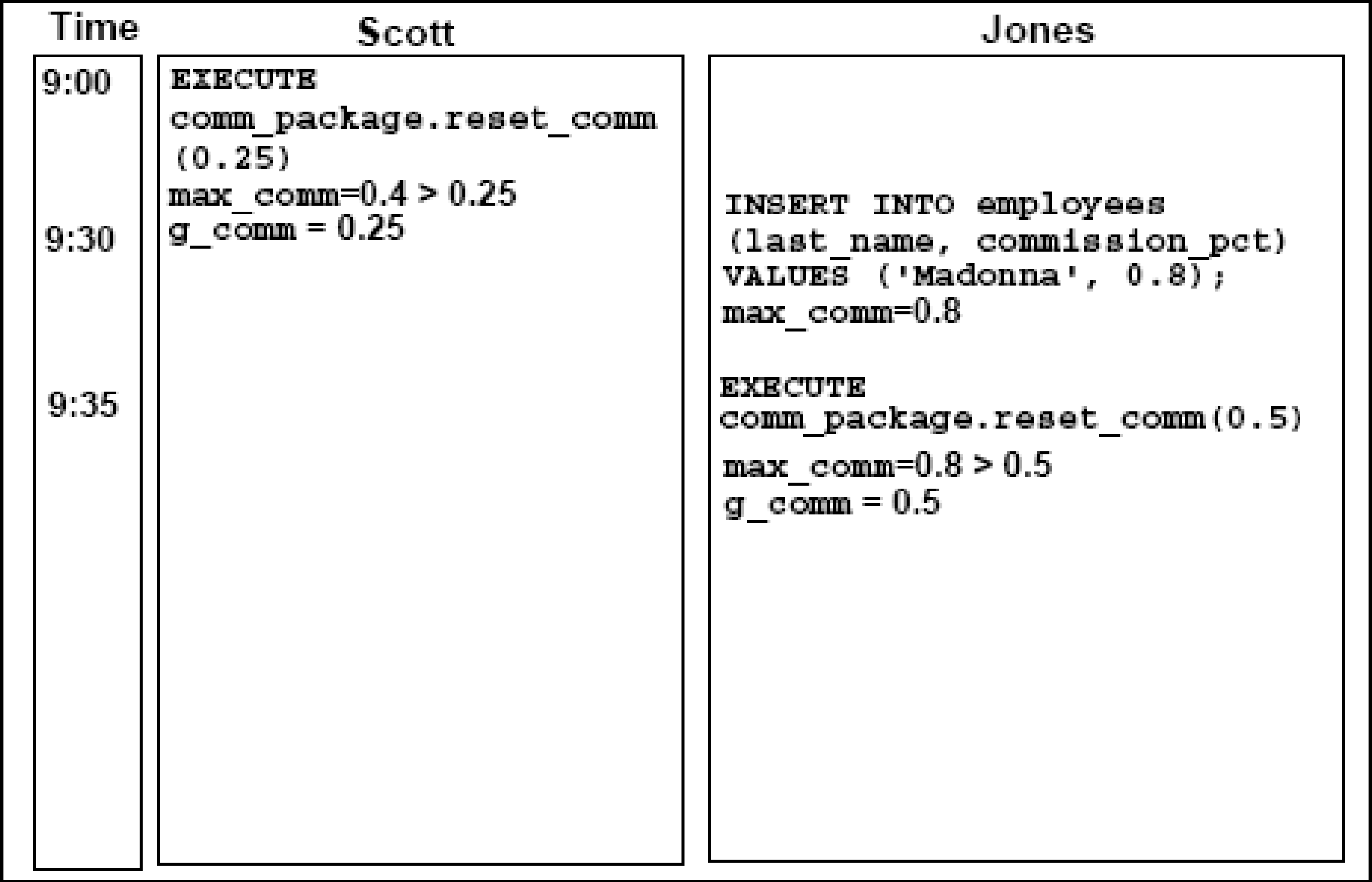
109 rows selected.

Persistent State of Package Variables: Example

```
CREATE OR REPLACE PACKAGE comm_package IS
  g_comm  NUMBER := 10;              --initialized to 10
  PROCEDURE reset_comm (p_comm   IN   NUMBER);
END comm_package;
/
```

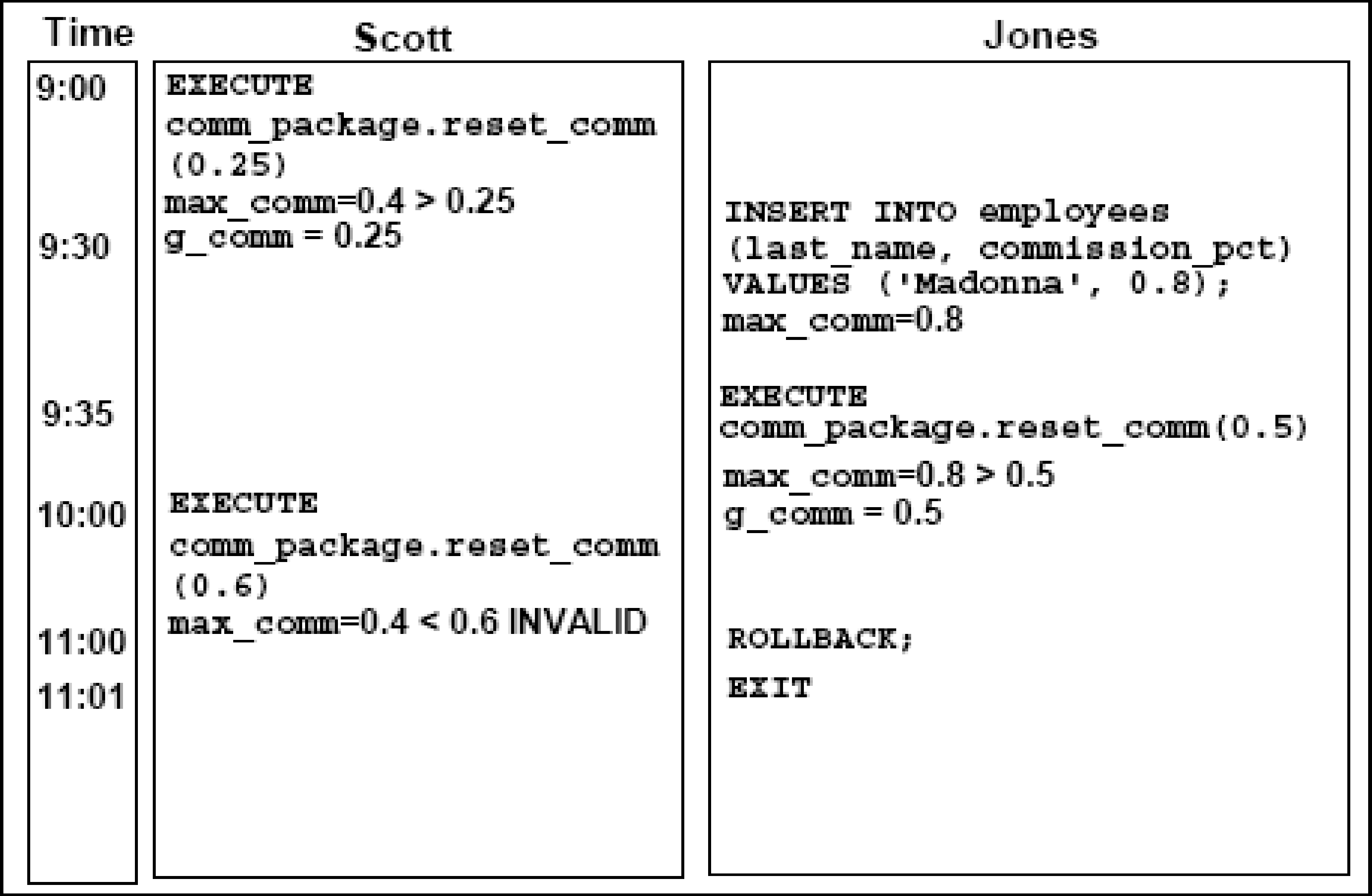
```
CREATE OR REPLACE PACKAGE BODY comm_package IS
   FUNCTION  validate_comm (p_comm  IN  NUMBER)
             RETURN BOOLEAN
   IS v_max_comm    NUMBER;
   BEGIN
    ...        -- validates commission to be less than maximum
               --  commission in the table
   END validate_comm;
   PROCEDURE  reset_comm (p_comm   IN  NUMBER)
   IS BEGIN
     ...       -- calls validate_comm with specified value
   END reset_comm;
END comm_package;
/
```

Persistent State of Package Variables



Time	Scott	Jones
9:00	`EXECUTE comm_package.reset_comm (0.25)` max_comm=0.4 > 0.25 g_comm = 0.25	
9:30		`INSERT INTO employees (last_name, commission_pct) VALUES ('Madonna', 0.8);` max_comm=0.8
9:35		`EXECUTE comm_package.reset_comm(0.5)` max_comm=0.8 > 0.5 g_comm = 0.5

Persistent State of Package Variables



Time	Scott	Jones

9:00	`EXECUTE comm_package.reset_comm (0.25)`
	max_comm=0.4 > 0.25
	g_comm = 0.25	

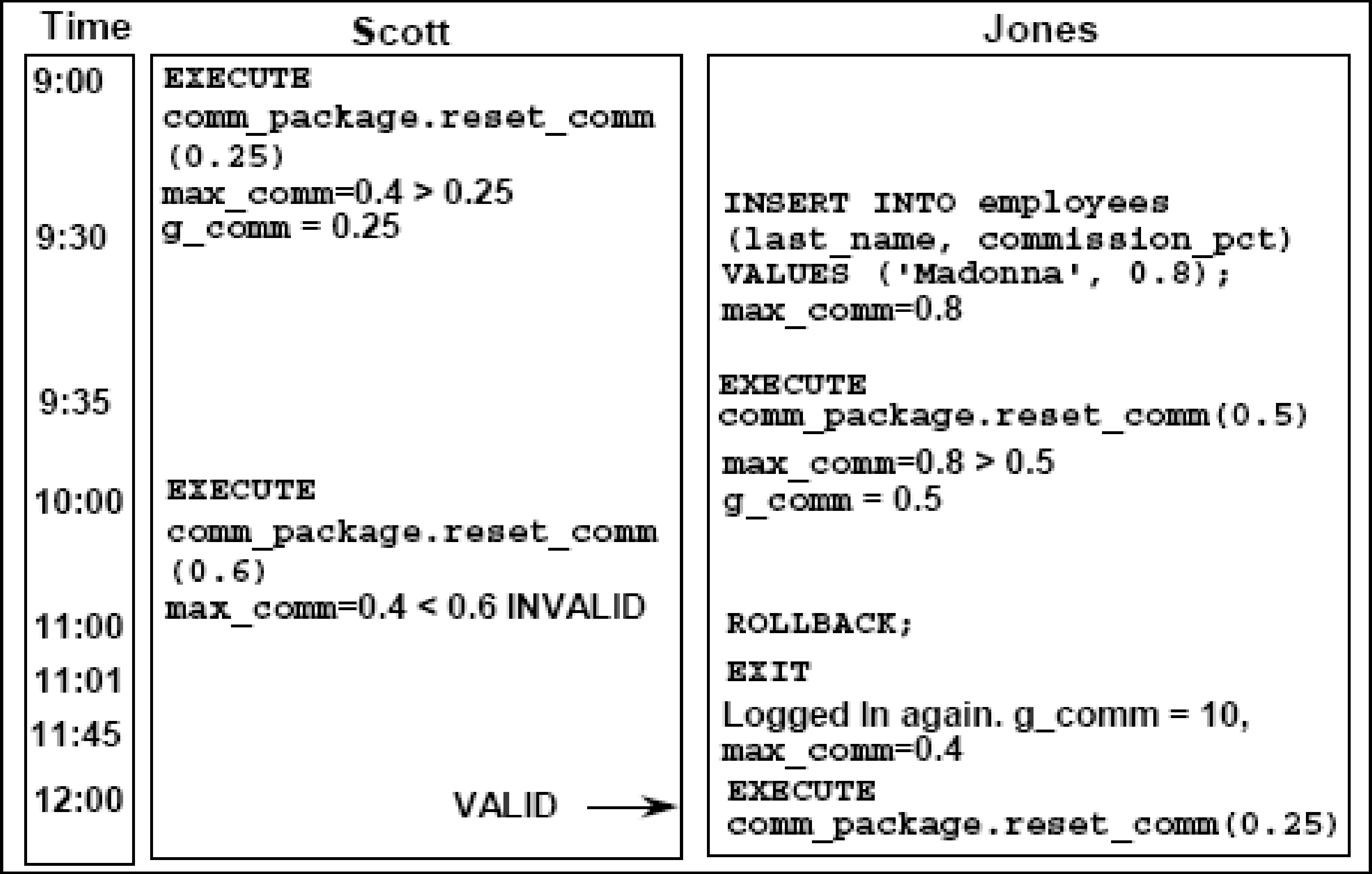
9:30		`INSERT INTO employees (last_name, commission_pct) VALUES ('Madonna', 0.8);`
		max_comm=0.8

9:35		`EXECUTE comm_package.reset_comm(0.5)`
		max_comm=0.8 > 0.5
		g_comm = 0.5

10:00	`EXECUTE comm_package.reset_comm (0.6)`
	max_comm=0.4 < 0.6 INVALID	

11:00		`ROLLBACK;`
11:01		`EXIT`

Persistent State of Package Variables



Time	Scott	Jones
9:00	EXECUTE comm_package.reset_comm (0.25) max_comm=0.4 > 0.25 g_comm = 0.25	
9:30		INSERT INTO employees (last_name, commission_pct) VALUES ('Madonna', 0.8); max_comm=0.8
9:35		EXECUTE comm_package.reset_comm(0.5) max_comm=0.8 > 0.5 g_comm = 0.5
10:00	EXECUTE comm_package.reset_comm (0.6) max_comm=0.4 < 0.6 INVALID	
11:00		ROLLBACK;
11:01		EXIT
11:45		Logged In again. g_comm = 10, max_comm=0.4
12:00	VALID →	EXECUTE comm_package.reset_comm(0.25)

Controlling the Persistent State of a Package Cursor

Example:

```
CREATE OR REPLACE PACKAGE pack_cur
IS
  CURSOR c1 IS SELECT employee_id
                 FROM employees
                 ORDER BY employee_id DESC;
  PROCEDURE proc1_3rows;
  PROCEDURE proc4_6rows;
END pack_cur;
/
```

Package created

Controlling the Persistent State of a Package Cursor

```
CREATE OR REPLACE PACKAGE BODY pack_cur IS
  v_empno NUMBER;
  PROCEDURE proc1_3rows IS
  BEGIN
          OPEN c1;
          LOOP
            FETCH c1 INTO v_empno;
            DBMS_OUTPUT.PUT_LINE('Id :' ||(v_empno));
            EXIT WHEN c1%ROWCOUNT >= 3;
          END LOOP;
  END proc1_3rows;
  PROCEDURE proc4_6rows IS
  BEGIN
          LOOP
            FETCH c1 INTO v_empno;
            DBMS_OUTPUT.PUT_LINE('Id :' ||(v_empno));
            EXIT WHEN c1%ROWCOUNT >= 6;
          END LOOP;
          CLOSE c1;
  END proc4_6rows;
END pack_cur;
/
```

Executing PACK_CUR

```
SET SERVEROUTPUT ON
EXECUTE pack_cur.proc1_3rows
EXECUTE pack_cur.proc4_6rows
```

```
Id :208
Id :207
Id :206
PL/SQL procedure successfully completed
Id :205
Id :204
Id :203
PL/SQL procedure successfully completed
```

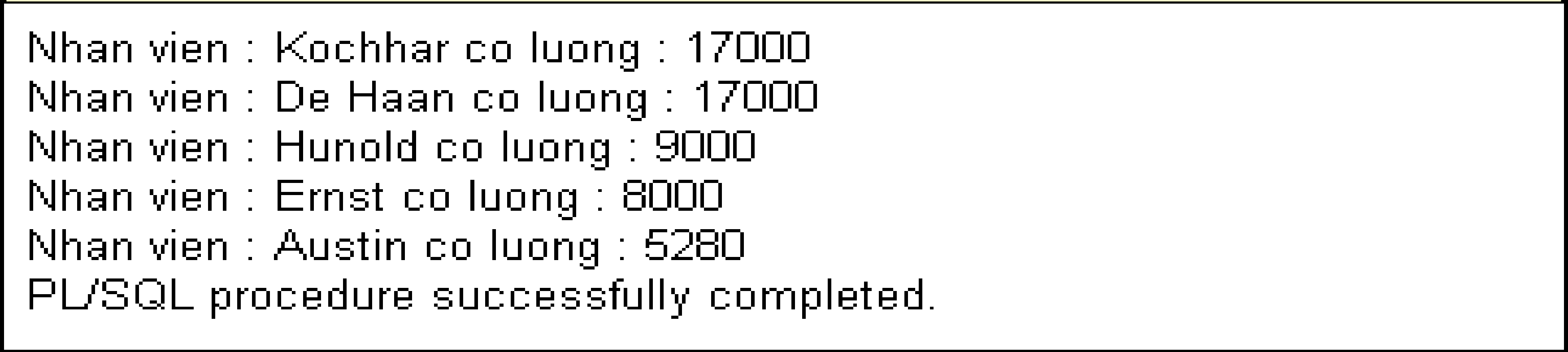
PL/SQL Tables and Records in Packages

```
CREATE OR REPLACE PACKAGE emp_package IS
       TYPE emp_table_type IS TABLE OF employees%ROWTYPE
                                     INDEX BY BINARY_INTEGER;
       PROCEDURE read_emp_table (p_emp_table OUT emp_table_type);
END emp_package;
/
CREATE OR REPLACE PACKAGE BODY emp_package IS
     PROCEDURE read_emp_table (p_emp_table OUT emp_table_type) IS
    i BINARY_INTEGER := 0;
  BEGIN
    FOR emp_record IN (SELECT * FROM employees)  LOOP
           p_emp_table(i) := emp_record;
           i:= i+1;
     END LOOP;
   END read_emp_table;
END emp_package;
/
```

ORACLE

PL/SQL Tables and Records in Packages

```
SET SERVEROUTPUT ON
DECLARE
   v_emp_table emp_package.emp_table_type;
BEGIN
   emp_package.read_emp_table(v_emp_table);
   FOR i IN 1 .. 5 LOOP
     DBMS_OUTPUT.PUT_LINE(‘Nhan vien : '||v_emp_table(i).last_name ||
                                              ‘ co luong : ‘|| v_emp_table(i).salary);
   END loop;
END;
/
```



```
Nhan vien : Kochhar co luong : 17000
Nhan vien : De Haan co luong : 17000
Nhan vien : Hunold co luong : 9000
Nhan vien : Ernst co luong : 8000
Nhan vien : Austin co luong : 5280
PL/SQL procedure successfully completed.
```

Summary

In this lesson, you should have learned how to:

- Overload subprograms
- Use forward referencing
- Use one-time-only procedures
- Describe the purity level of package functions
- Identify the persistent state of packaged objects

Practice 13 Overview

This practice covers the following topics:

- Using overloaded subprograms
- Creating a one-time-only procedure