9

Creating Procedures

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- Distinguish anonymous PL/SQL blocks from named PL/SQL blocks (subprograms)
- Describe subprograms
- List the benefits of using subprograms
- List the different environments from which subprograms can be invoked

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- Describe PL/SQL blocks and subprograms
- Describe the uses of procedures
- Create procedures
- Differentiate between formal and actual parameters
- List the features of different parameter modes
- Create procedures with parameters
- Invoke a procedure
- Handle exceptions in procedures
- Remove a procedure

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview of Subprograms

A subprogram:

- Is a named PL/SQL block that can accept parameters and be invoked from a calling environment
- Is of two types:
  - A procedure that performs an action
  - A function that computes a value
- Is based on standard PL/SQL block structure
- Provides modularity, reusability, extensibility, and maintainability
- Provides easy maintenance, improved data security and integrity, improved performance, and improved code clarity

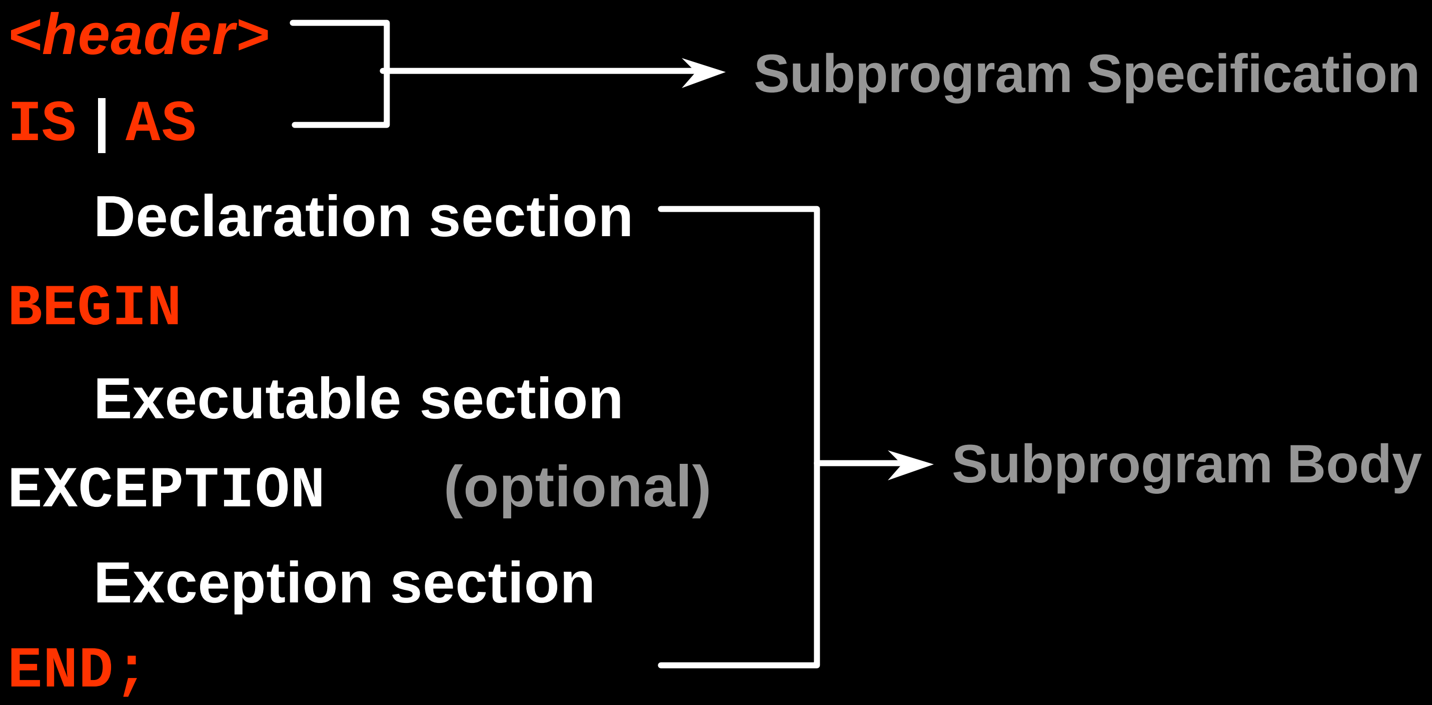
Copyright © Oracle Corporation, 2001. All rights reserved.

Block Structure for Anonymous PL/SQL Blocks

```
DECLARE      (optional)
             Declare PL/SQL objects to be used
             within this block
BEGIN        (mandatory)
             Define the executable statements
EXCEPTION    (optional)
             Define the actions that take place if
             an error or exception arises
END;         (mandatory)
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Block Structure for PL/SQL Subprograms



```
<header>
IS | AS
    Declaration section
BEGIN
    Executable section
EXCEPTION      (optional)
    Exception section
END;
```

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Benefits of Subprograms

- Easy maintenance
- Improved data security and integrity
- Improved performance
- Improved code clarity

Copyright © Oracle Corporation, 2001. All rights reserved.

What Is a Procedure?

- A procedure is a type of subprogram that performs an action.
- A procedure can be stored in the database, as a schema object, for repeated execution.

Copyright © Oracle Corporation, 2001. All rights reserved.

Syntax for Creating Procedures

```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [(parameter1 [mode1] datatype1,
  parameter2 [mode2] datatype2,
  . . .)]
IS|AS
PL/SQL Block;
```

- The `REPLACE` option indicates that if the procedure exists, it will be dropped and replaced with the new version created by the statement.
- PL/SQL block starts with either `BEGIN` or the declaration of local variables and ends with either `END` or `END` *procedure_name.*

Copyright © Oracle Corporation, 2001. All rights reserved.

Formal Versus Actual Parameters

- Formal parameters: variables declared in the parameter list of a subprogram specification

  Example:

```
CREATE PROCEDURE raise_sal(p_id NUMBER, p_amount NUMBER)
...
END raise_sal;
```

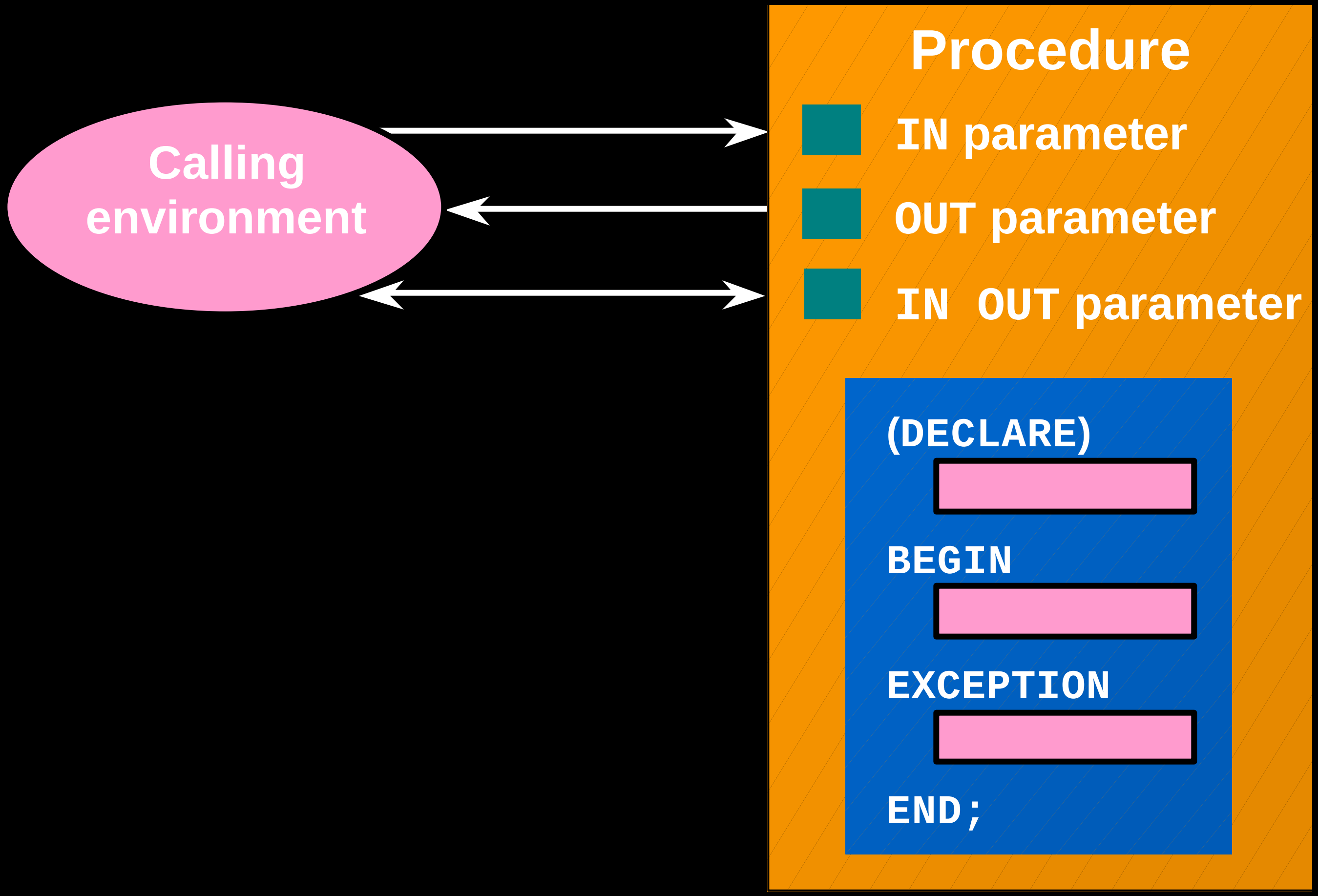
- Actual parameters: variables or expressions referenced in the parameter list of a subprogram call

  Example:

```
raise_sal(v_id, 2000)
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Procedural Parameter Modes



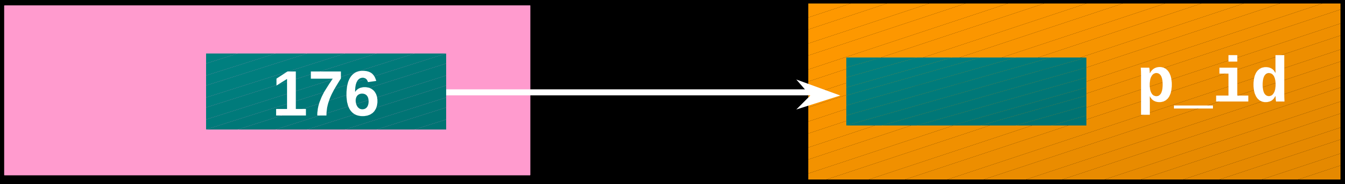
Copyright © Oracle Corporation, 2001. All rights reserved.

Creating Procedures with Parameters

IN	OUT	IN OUT
Default mode	Must be specified	Must be specified
Value is passed into subprogram	Returned to calling environment	Passed into subprogram; returned to calling environment
Formal parameter acts as a constant	Uninitialized variable	Initialized variable
Actual parameter can be a literal, expression, constant, or initialized variable	Must be a variable	Must be a variable
Can be assigned a default value	Cannot be assigned a default value	Cannot be assigned a default value

Copyright © Oracle Corporation, 2001. All rights reserved.

IN Parameters: Example

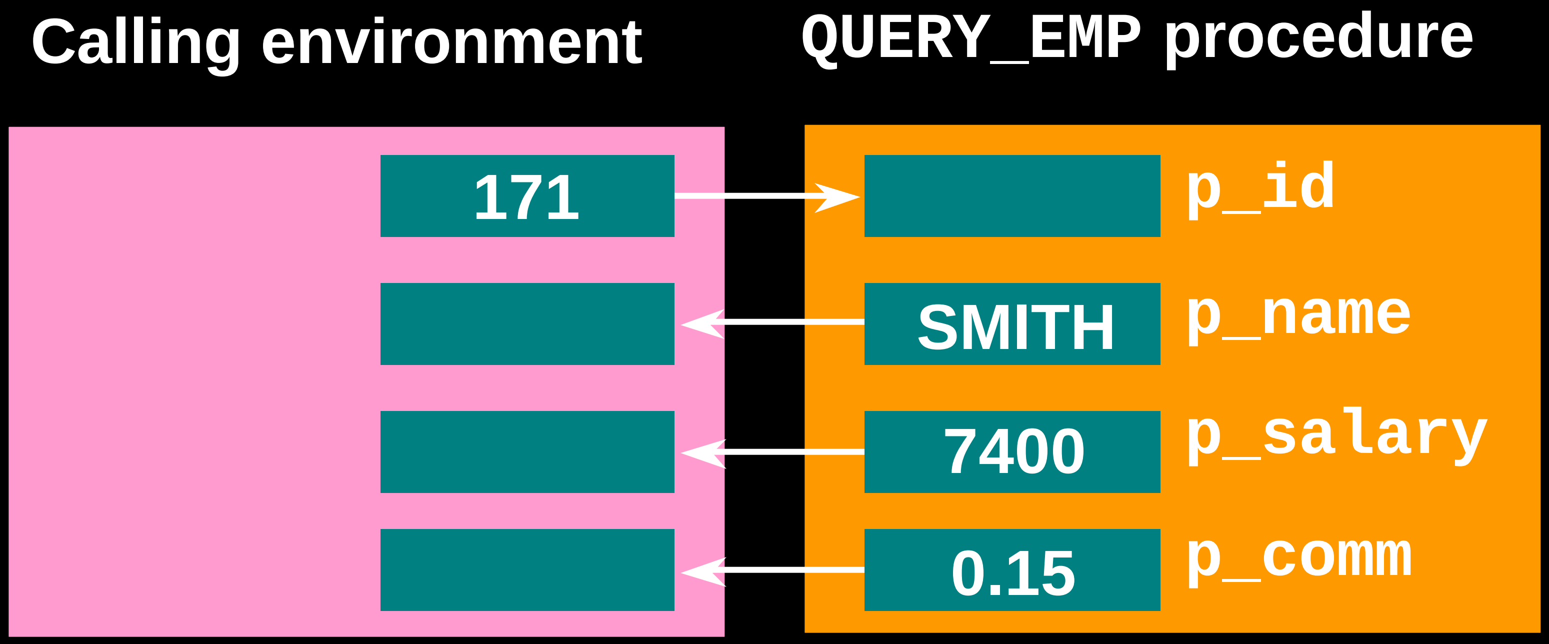


```
CREATE OR REPLACE PROCEDURE raise_salary
  (p_id IN employees.employee_id%TYPE)
IS
BEGIN
  UPDATE employees
  SET    salary = salary * 1.10
  WHERE  employee_id = p_id;
END raise_salary;
/
```

```
Procedure created.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

OUT Parameters: Example



Copyright © Oracle Corporation, 2001. All rights reserved.

OUT Parameters: Example

emp_query.sql

```
CREATE OR REPLACE PROCEDURE query_emp
  (p_id       IN   employees.employee_id%TYPE,
   p_name     OUT  employees.last_name%TYPE,
   p_salary   OUT  employees.salary%TYPE,
   p_comm     OUT  employees.commission_pct%TYPE)
IS
BEGIN
  SELECT   last_name, salary, commission_pct
   INTO    p_name, p_salary, p_comm
   FROM    employees
   WHERE   employee_id = p_id;
END query_emp;
/
```

```
Procedure created.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

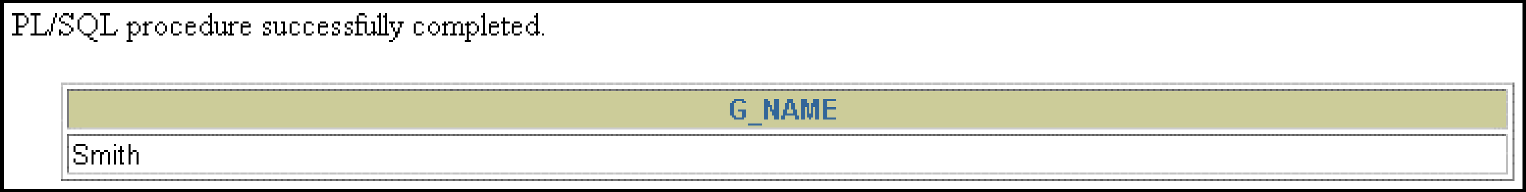
Viewing OUT Parameters

- Load and run the `emp_query.sql` script file to create the `QUERY_EMP` procedure.
- Declare host variables, execute the `QUERY_EMP` procedure, and print the value of the global `G_NAME` variable.

```
VARIABLE g_name     VARCHAR2(25)
VARIABLE g_sal      NUMBER
VARIABLE g_comm     NUMBER

EXECUTE query_emp(171, :g_name, :g_sal, :g_comm)

PRINT g_name
```

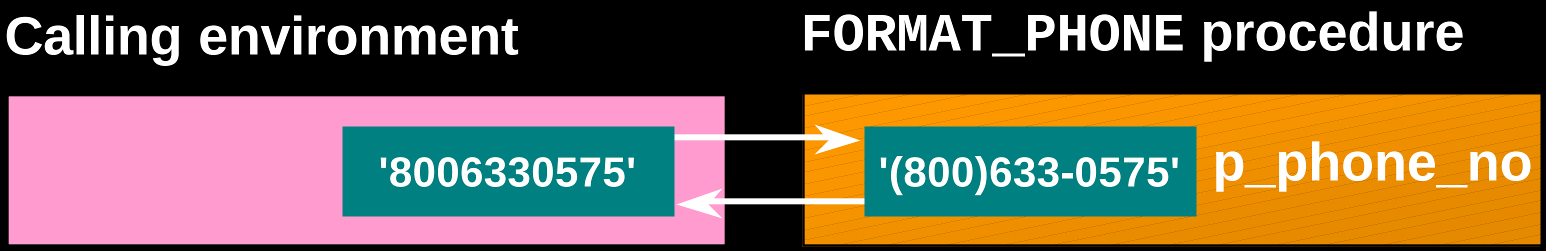


```
PL/SQL procedure successfully completed.

G_NAME
Smith
```

Copyright © Oracle Corporation, 2001. All rights reserved.

IN OUT Parameters



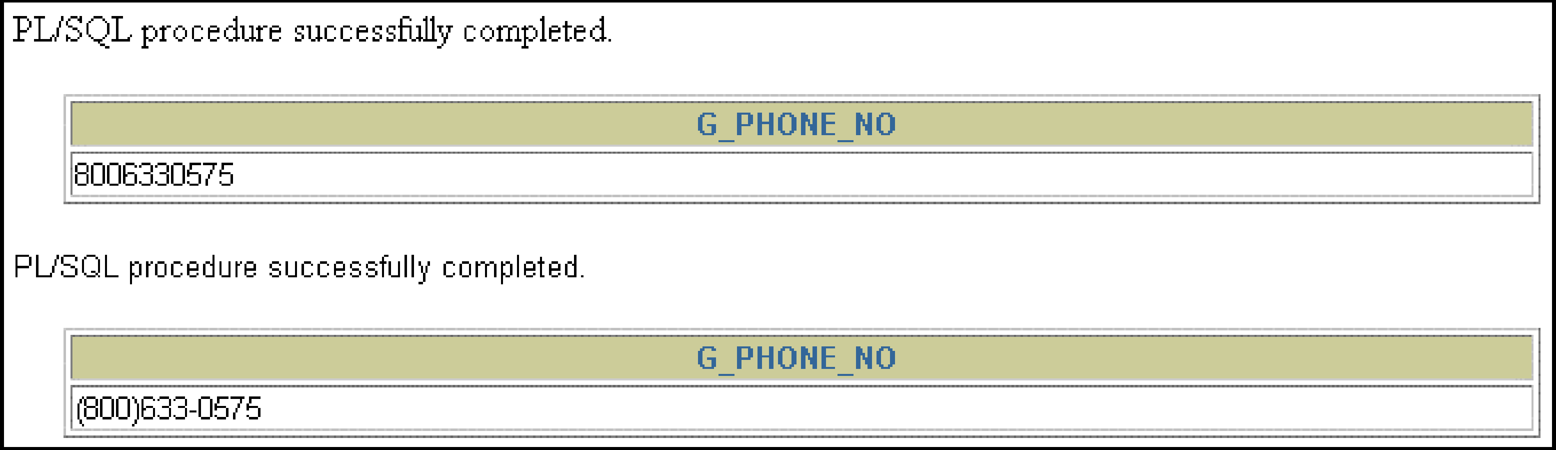
```
CREATE OR REPLACE PROCEDURE format_phone
  (p_phone_no IN OUT VARCHAR2)
IS
BEGIN
  p_phone_no := '(' || SUBSTR(p_phone_no,1,3) ||
                ')' || SUBSTR(p_phone_no,4,3) ||
                '-' || SUBSTR(p_phone_no,7);
END format_phone;
/
```

```
Procedure created.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Viewing IN OUT Parameters

```
VARIABLE g_phone_no VARCHAR2(15)
BEGIN
  :g_phone_no := '8006330575';
END;
/
PRINT g_phone_no
EXECUTE format_phone (:g_phone_no)
PRINT g_phone_no
```



```
PL/SQL procedure successfully completed.

      G_PHONE_NO
8006330575

PL/SQL procedure successfully completed.

      G_PHONE_NO
(800)633-0575
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Methods for Passing Parameters

- Positional: List actual parameters in the same order as formal parameters.
- Named: List actual parameters in arbitrary order by associating each with its corresponding formal parameter.
- Combination: List some of the actual parameters as positional and some as named.

Copyright © Oracle Corporation, 2001. All rights reserved.

DEFAULT Option for Parameters

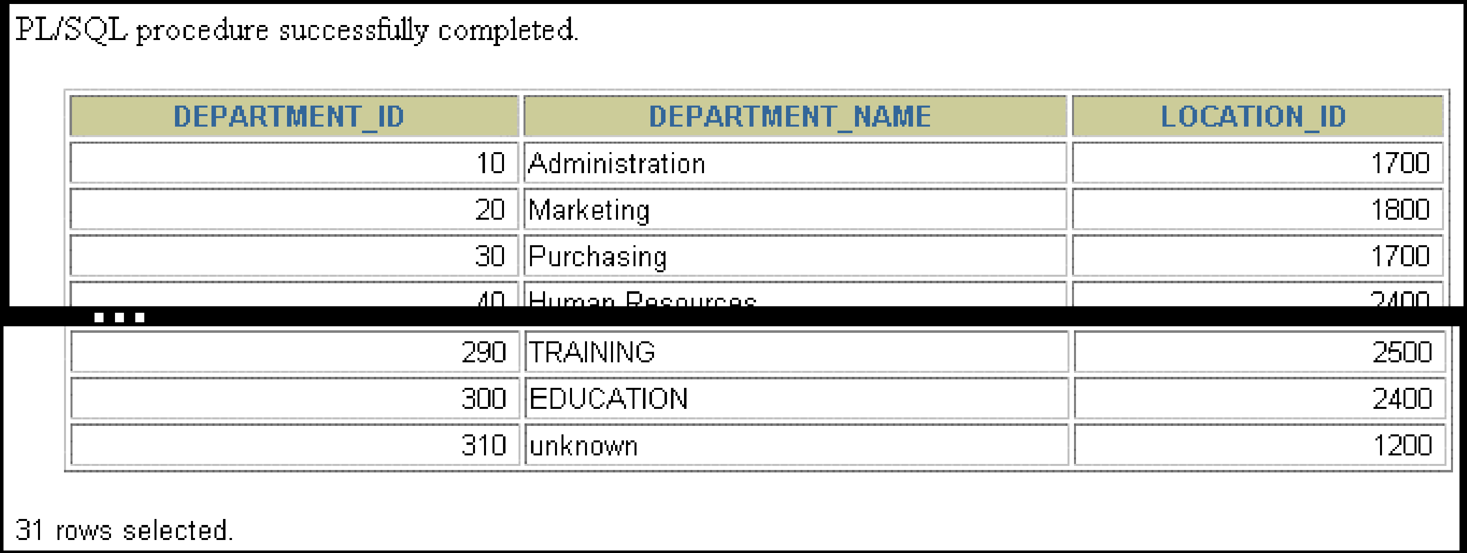
```
CREATE OR REPLACE PROCEDURE add_dept
  (p_name  IN departments.department_name%TYPE
                              DEFAULT 'unknown',
   p_loc   IN departments.location_id%TYPE
                              DEFAULT 1700)
IS
BEGIN
  INSERT INTO departments(department_id,
            department_name, location_id)
  VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_dept;
/
```

```
Procedure created.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Examples of Passing Parameters

```
BEGIN
  add_dept;
  add_dept ('TRAINING', 2500);
  add_dept ( p_loc => 2400, p_name =>'EDUCATION');
  add_dept ( p_loc => 1200) ;
END;
/
SELECT department_id, department_name, location_id
FROM departments;
```



```
PL/SQL procedure successfully completed.

DEPARTMENT_ID  DEPARTMENT_NAME   LOCATION_ID
           10  Administration           1700
           20  Marketing                1800
           30  Purchasing               1700
           40  Human Resources          2400
...
          290  TRAINING                 2500
          300  EDUCATION                2400
          310  unknown                  1200

31 rows selected.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Declaring Subprograms

leave_emp2.sql

```
CREATE OR REPLACE PROCEDURE leave_emp2
  (p_id  IN  employees.employee_id%TYPE)
IS
  PROCEDURE log_exec
  IS
  BEGIN
    INSERT INTO log_table (user_id, log_date)
    VALUES (USER, SYSDATE);
  END log_exec;
BEGIN
  DELETE FROM employees
  WHERE employee_id = p_id;
  log_exec;
END leave_emp2;
/
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking a Procedure from an Anonymous PL/SQL Block

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

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking a Procedure from Another Procedure

process_emps.sql

```
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;
/
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Removing Procedures

Drop a procedure stored in the database.

Syntax:

```
DROP PROCEDURE procedure_name
```

Example:

```
DROP PROCEDURE raise_salary;
```

```
Procedure dropped.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned that:

- A procedure is a subprogram that performs an action.
- You create procedures by using the `CREATE PROCEDURE` command.
- You can compile and save a procedure in the database.
- Parameters are used to pass data from the calling environment to the procedure.
- There are three parameter modes: `IN`, `OUT`, and `IN OUT`.

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

- Local subprograms are programs that are defined within the declaration section of another program.
- Procedures can be invoked from any tool or language that supports PL/SQL.
- You should be aware of the effect of handled and unhandled exceptions on transactions and calling procedures.
- You can remove procedures from the database by using the `DROP PROCEDURE` command.
- Procedures can serve as building blocks for an application.

Copyright © Oracle Corporation, 2001. All rights reserved.

10

Creating Functions

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- Describe the uses of functions
- Create stored functions
- Invoke a function
- Remove a function
- Differentiate between a procedure and a function

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview of Stored Functions

- A function is a named PL/SQL block that returns a value.
- A function can be stored in the database as a schema object for repeated execution.
- A function is called as part of an expression.

Copyright © Oracle Corporation, 2001. All rights reserved.

Syntax for Creating Functions

```
CREATE [OR REPLACE] FUNCTION function_name
 [(parameter1 [mode1] datatype1,
  parameter2 [mode2] datatype2,
  . . .)]
RETURN datatype
IS|AS
PL/SQL Block;
```

The PL/SQL block must have at least one RETURN statement.

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Stored Function by Using *i*SQL*Plus

1. Enter the text of the `CREATE FUNCTION` statement in an editor and save it as a SQL script file.
2. Run the script file to store the source code and compile the function.
3. Use `SHOW ERRORS` to see compilation errors.
4. When successfully compiled, invoke the function.

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Stored Function by Using *i*SQL*Plus: Example

get_salary.sql

```
CREATE OR REPLACE FUNCTION get_sal
      (p_id  IN employees.employee_id%TYPE)
      RETURN NUMBER
IS
      v_salary employees.salary%TYPE :=0;
BEGIN
      SELECT salary
      INTO   v_salary
      FROM   employees
      WHERE  employee_id = p_id;
     RETURN v_salary;
END get_sal;
/
```

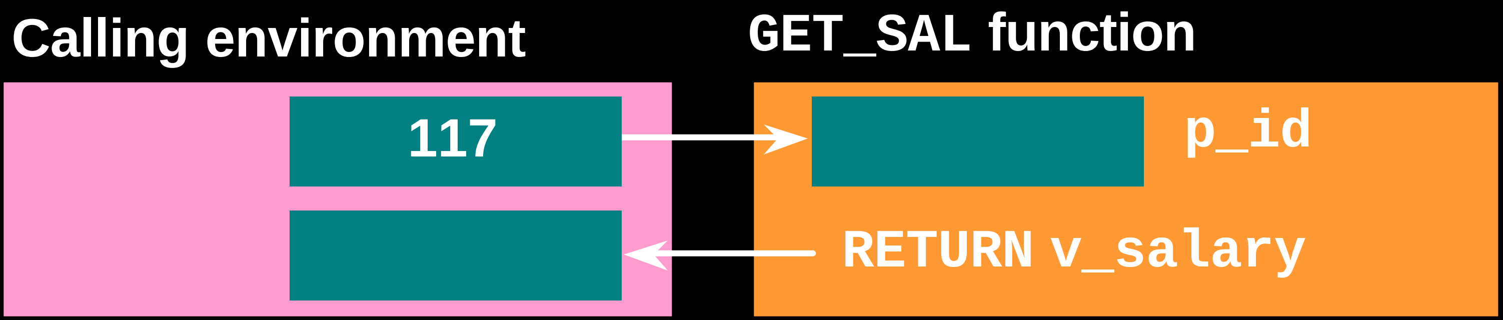
Copyright © Oracle Corporation, 2001. All rights reserved.

Executing Functions

- Invoke a function as part of a PL/SQL expression.
- Create a variable to hold the returned value.
- Execute the function. The variable will be populated by the value returned through a `RETURN` statement.

Copyright © Oracle Corporation, 2001. All rights reserved.

Executing Functions: Example



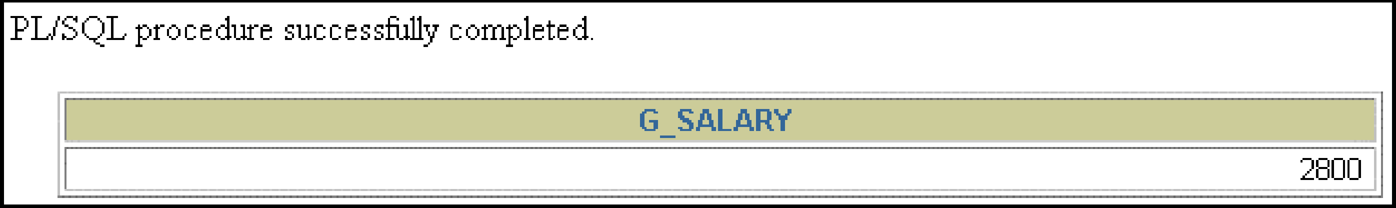
1. Load and run the `get_salary.sql` file to create the function

```
VARIABLE g_salary NUMBER

EXECUTE :g_salary := get_sal(117)

PRINT g_salary
```

Steps marked 2, 3 and 4 point to the three lines above.



```
PL/SQL procedure successfully completed.

G_SALARY
    2800
```

Copyright © Oracle Corporation, 2001. All rights reserved.

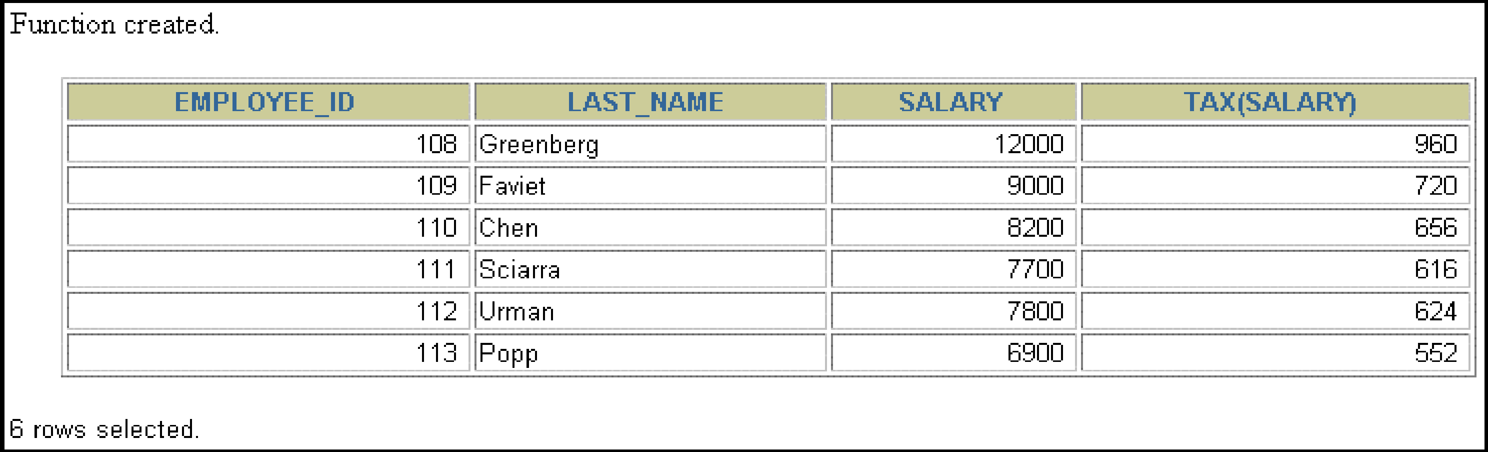
Advantages of User-Defined Functions in SQL Expressions

- Extend SQL where activities are too complex, too awkward, or unavailable with SQL
- Can increase efficiency when used in the `WHERE` clause to filter data, as opposed to filtering the data in the application
- Can manipulate character strings

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking Functions in SQL Expressions: Example

```
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.

Copyright © Oracle Corporation, 2001. All rights reserved.

Restrictions on Calling Functions from SQL Expressions

To be callable from SQL expressions, a user-defined function must:

- Be a stored function
- Accept only `IN` parameters
- Accept only valid SQL data types, not PL/SQL specific types, as parameters
- Return data types that are valid SQL data types, not PL/SQL specific types

Copyright © Oracle Corporation, 2001. All rights reserved.

Restrictions on Calling Functions from SQL Expressions

- Functions called from SQL expressions cannot contain DML statements.
- Functions called from UPDATE/DELETE statements on a table T cannot contain DML on the same table T.
- Functions called from an UPDATE or a DELETE statement on a table T cannot query the same table.
- Functions called from SQL statements cannot contain statements that end the transactions.
- Calls to subprograms that break the previous restriction are not allowed in the function.

Copyright © Oracle Corporation, 2001. All rights reserved.

Restrictions on Calling from SQL

```
CREATE OR REPLACE FUNCTION dml_call_sql (p_sal NUMBER)
   RETURN NUMBER IS
BEGIN
   INSERT INTO employees(employee_id, last_name, email,
                         hire_date, job_id, salary)
         VALUES(1, 'employee 1', 'emp1@company.com',
                    SYSDATE, 'SA_MAN', 1000);
   RETURN (p_sal + 100);
END;
/
```

```
Function created.
```

```
UPDATE employees SET salary = dml_call_sql(2000)
 WHERE employee_id = 170;
```

```
UPDATE employees SET salary = dml_call_sql(2000)
                              *
ERROR at line 1:
ORA-04091: table PLSQL.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "PLSQL.DML_CALL_SQL", line 4
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Removing Functions

Drop a stored function.

Syntax:

```
DROP FUNCTION function_name
```

Example:

```
DROP FUNCTION get_sal;
```

```
Function dropped.
```

- All the privileges granted on a function are revoked when the function is dropped.
- The `CREATE OR REPLACE` syntax is equivalent to dropping a function and recreating it. Privileges granted on the function remain the same when this syntax is used.

Copyright © Oracle Corporation, 2001. All rights reserved.

Comparing Procedures and Functions

Procedures	Functions
Execute as a PL/SQL statement	Invoke as part of an expression
Do not contain RETURN clause in the header	Must contain a RETURN clause in the header
Can return none, one, or many values	Must return a single value
Can contain a RETURN statement	Must contain at least one RETURN statement

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned that:

- A function is a named PL/SQL block that must return a value.
- A function is created by using the `CREATE FUNCTION` syntax.
- A function is invoked as part of an expression.
- A function stored in the database can be called in SQL statements.
- A function can be removed from the database by using the `DROP  FUNCTION` syntax.
- Generally, you use a procedure to perform an action and a function to compute a value.

ORACLE
Copyright © Oracle Corporation, 2001. All rights reserved.

11

Managing Subprograms

Copyright © Oracle Corporation, 2001. All rights reserved.

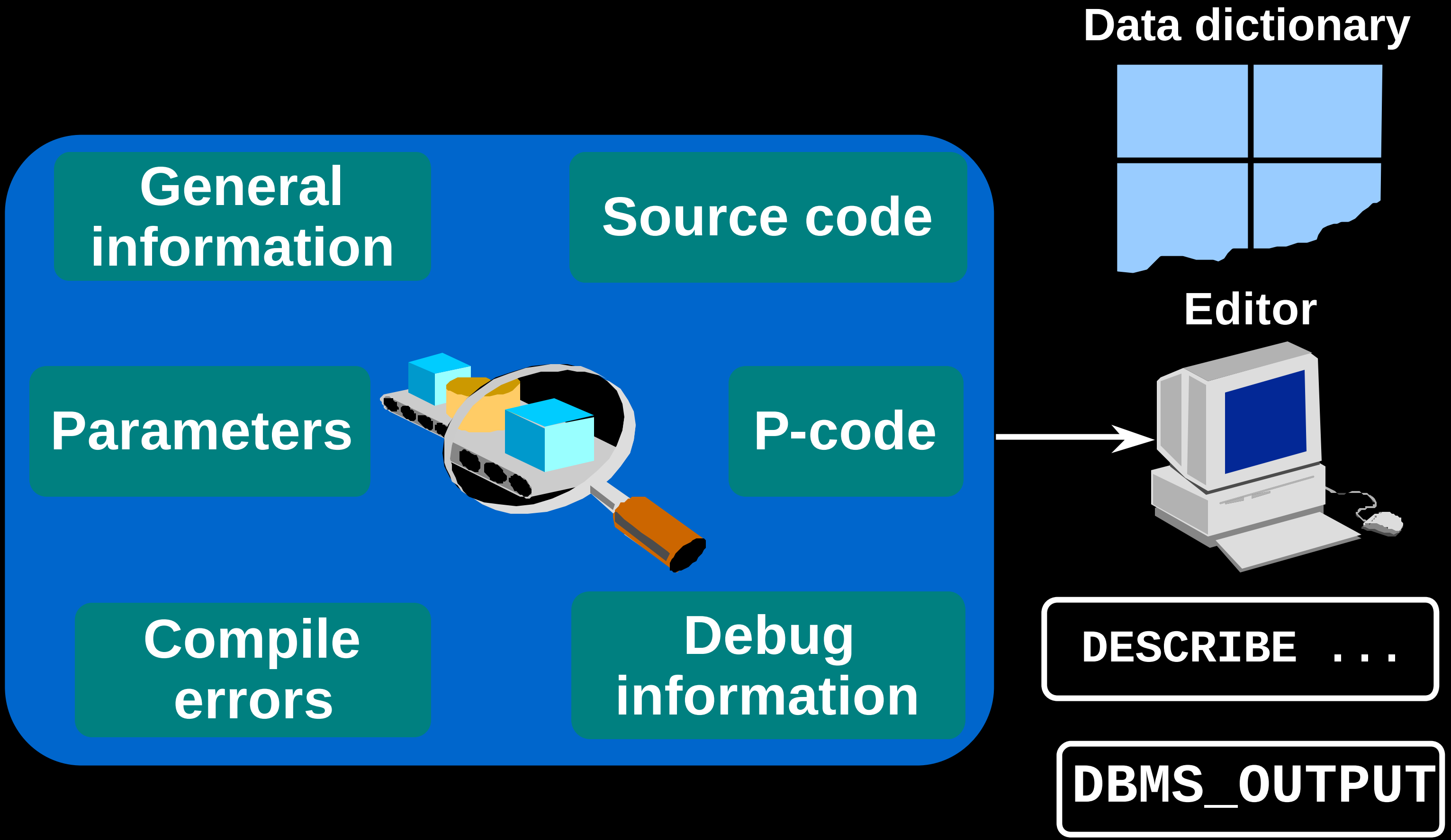
Objectives

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

- Contrast system privileges with object privileges
- Contrast invokers rights with definers rights
- Identify views in the data dictionary to manage stored objects
- Describe how to debug subprograms by using the `DBMS_OUTPUT` package

Copyright © Oracle Corporation, 2001. All rights reserved.

Managing Stored PL/SQL Objects



Copyright © Oracle Corporation, 2001. All rights reserved.

USER_OBJECTS

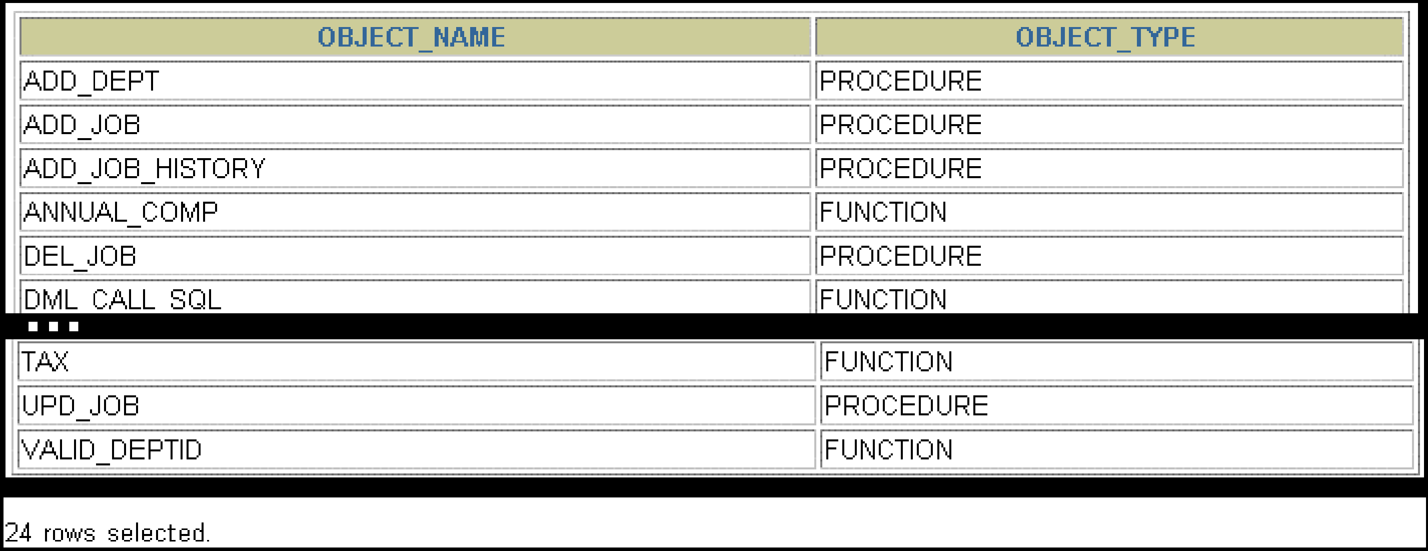
Column	Column Description
OBJECT_NAME	Name of the object
OBJECT_ID	Internal identifier for the object
OBJECT_TYPE	Type of object, for example, TABLE, PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER
CREATED	Date when the object was created
LAST_DDL_TIME	Date when the object was last modified
TIMESTAMP	Date and time when the object was last recompiled
STATUS	VALID or INVALID

*Abridged column list

Copyright © Oracle Corporation, 2001. All rights reserved.

List All Procedures and Functions

```
SELECT object_name, object_type
FROM  user_objects
WHERE object_type in ('PROCEDURE','FUNCTION')
ORDER BY object_name;
```



OBJECT_NAME	OBJECT_TYPE
ADD_DEPT	PROCEDURE
ADD_JOB	PROCEDURE
ADD_JOB_HISTORY	PROCEDURE
ANNUAL_COMP	FUNCTION
DEL_JOB	PROCEDURE
DML_CALL_SQL	FUNCTION
…	
TAX	FUNCTION
UPD_JOB	PROCEDURE
VALID_DEPTID	FUNCTION

```
24 rows selected.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

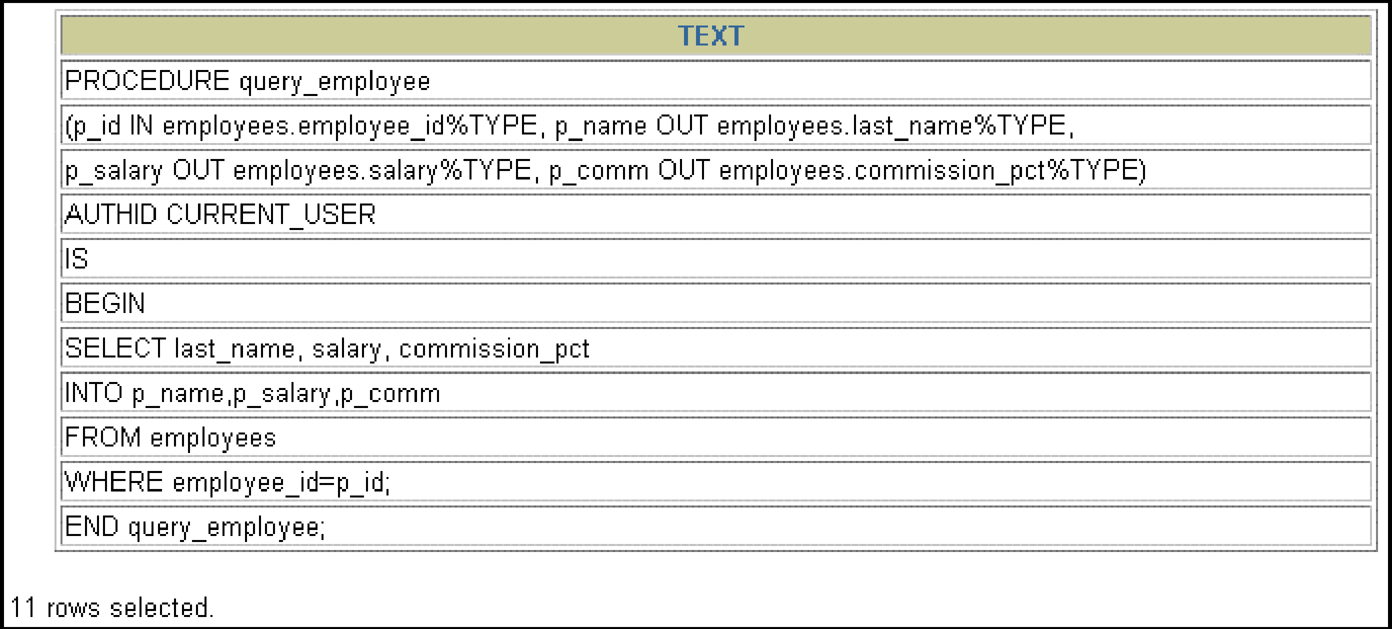
USER_SOURCE Data Dictionary View

Column	Column Description
`NAME`	Name of the object
`TYPE`	Type of object, for example, `PROCEDURE`, `FUNCTION`, `PACKAGE`, `PACKAGE BODY`
`LINE`	Line number of the source code
`TEXT`	Text of the source code line

Copyright © Oracle Corporation, 2001. All rights reserved.

List the Code of Procedures and Functions

```
SELECT  text
FROM  user_source
WHERE  name = 'QUERY_EMPLOYEE'
ORDER BY  line;
```



TEXT
PROCEDURE query_employee
(p_id IN employees.employee_id%TYPE, p_name OUT employees.last_name%TYPE,
p_salary OUT employees.salary%TYPE, p_comm OUT employees.commission_pct%TYPE)
AUTHID CURRENT_USER
IS
BEGIN
SELECT last_name, salary, commission_pct
INTO p_name,p_salary,p_comm
FROM employees
WHERE employee_id=p_id;
END query_employee;

11 rows selected.

Copyright © Oracle Corporation, 2001. All rights reserved.

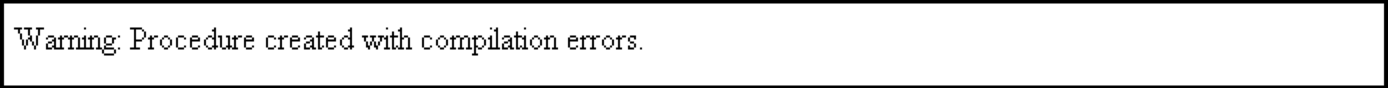
USER_ERRORS

Column	Column Description
`NAME`	Name of the object
`TYPE`	Type of object, for example, `PROCEDURE`, `FUNCTION`, `PACKAGE`, `PACKAGE BODY`, `TRIGGER`
`SEQUENCE`	Sequence number, for ordering
`LINE`	Line number of the source code at which the error occurs
`POSITION`	Position in the line at which the error occurs
`TEXT`	Text of the error message

Copyright © Oracle Corporation, 2001. All rights reserved.

Detecting Compilation Errors: Example

```
CREATE OR REPLACE PROCEDURE log_execution
IS
BEGIN
INPUT INTO log_table (user_id, log_date)
                                         -- wrong
VALUES (USER, SYSDATE);
END;
/
```

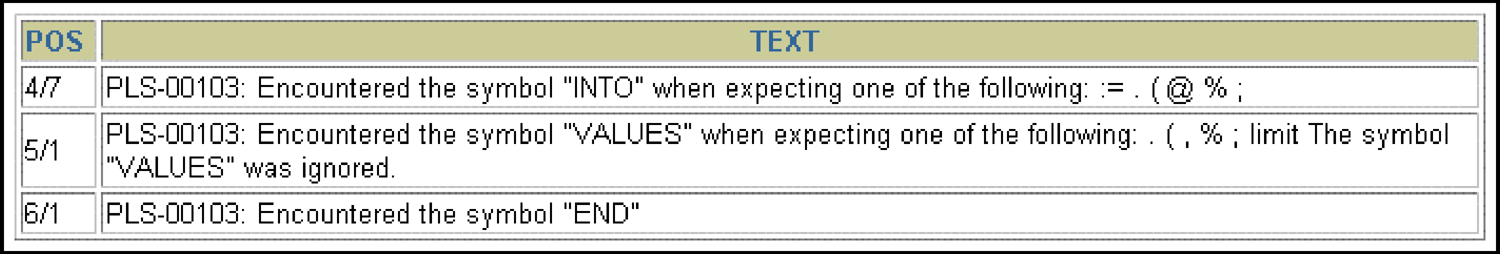


```
Warning: Procedure created with compilation errors.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

List Compilation Errors by Using USER_ERRORS

```
SELECT line || '/' || position POS, text
FROM   user_errors
WHERE  name = 'LOG_EXECUTION'
ORDER BY line;
```

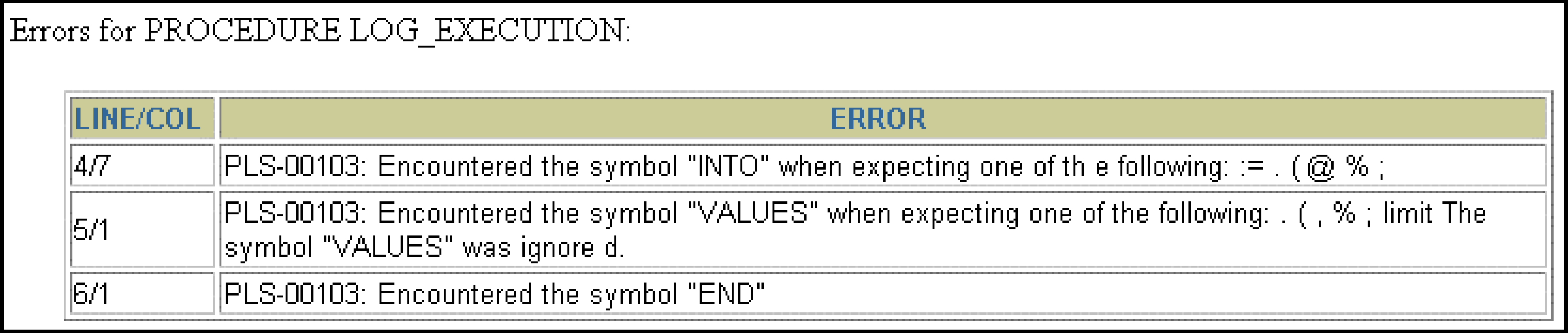


POS	TEXT
4/7	PLS-00103: Encountered the symbol "INTO" when expecting one of the following: := . (@ % ;
5/1	PLS-00103: Encountered the symbol "VALUES" when expecting one of the following: . (, % ; limit The symbol "VALUES" was ignored.
6/1	PLS-00103: Encountered the symbol "END"

Copyright © Oracle Corporation, 2001. All rights reserved.

List Compilation Errors by Using `SHOW ERRORS`

```
SHOW ERRORS PROCEDURE log_execution
```



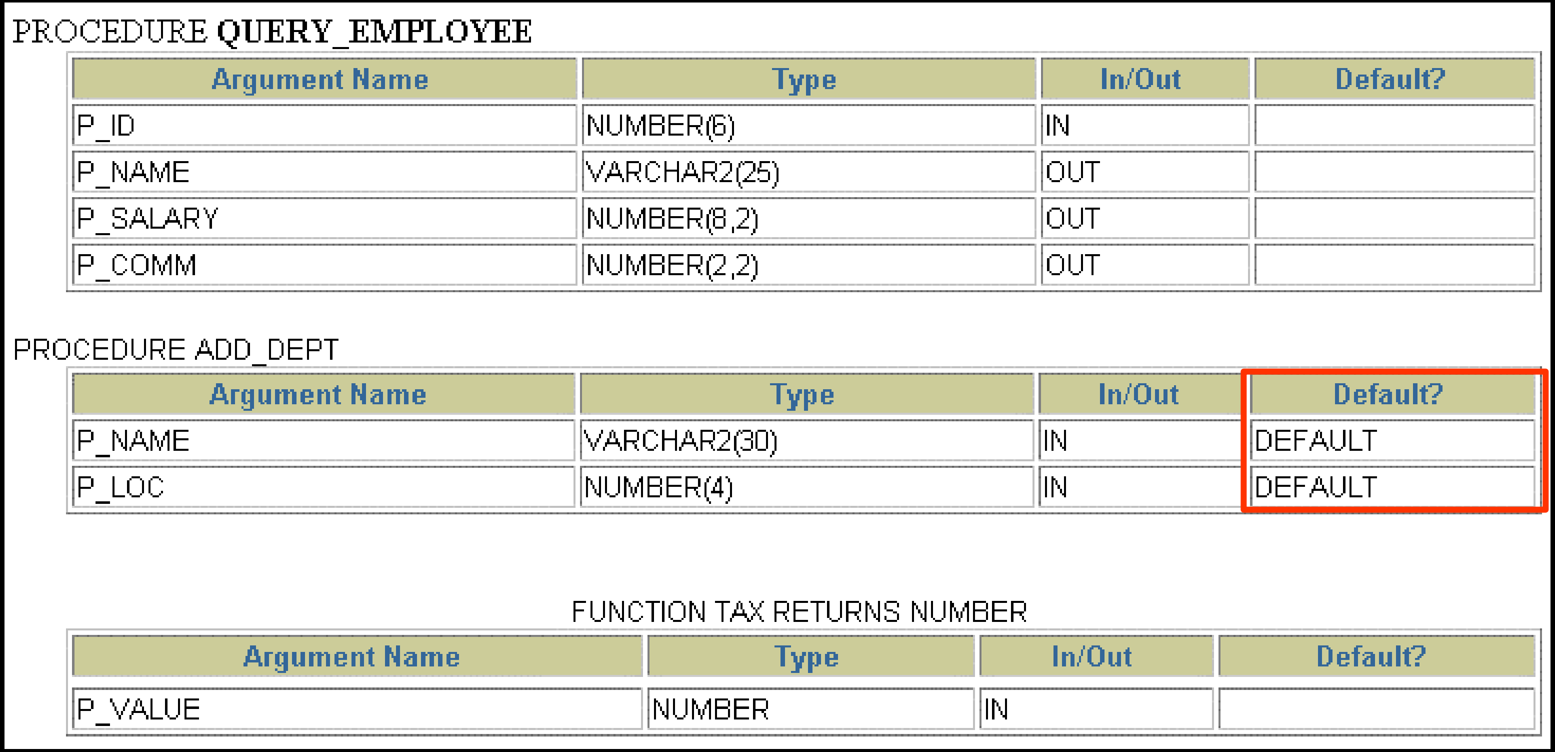
Errors for PROCEDURE LOG_EXECUTION:

LINE/COL	ERROR
4/7	PLS-00103: Encountered the symbol "INTO" when expecting one of th e following: := . (@ % ;
5/1	PLS-00103: Encountered the symbol "VALUES" when expecting one of the following: . (, % ; limit The symbol "VALUES" was ignore d.
6/1	PLS-00103: Encountered the symbol "END"

Copyright © Oracle Corporation, 2001. All rights reserved.

DESCRIBE in *i*SQL*Plus

```
DESCRIBE query_employee
DESCRIBE add_dept
DESCRIBE tax
```



PROCEDURE QUERY_EMPLOYEE

Argument Name	Type	In/Out	Default?
P_ID	NUMBER(6)	IN	
P_NAME	VARCHAR2(25)	OUT	
P_SALARY	NUMBER(8,2)	OUT	
P_COMM	NUMBER(2,2)	OUT	

PROCEDURE ADD_DEPT

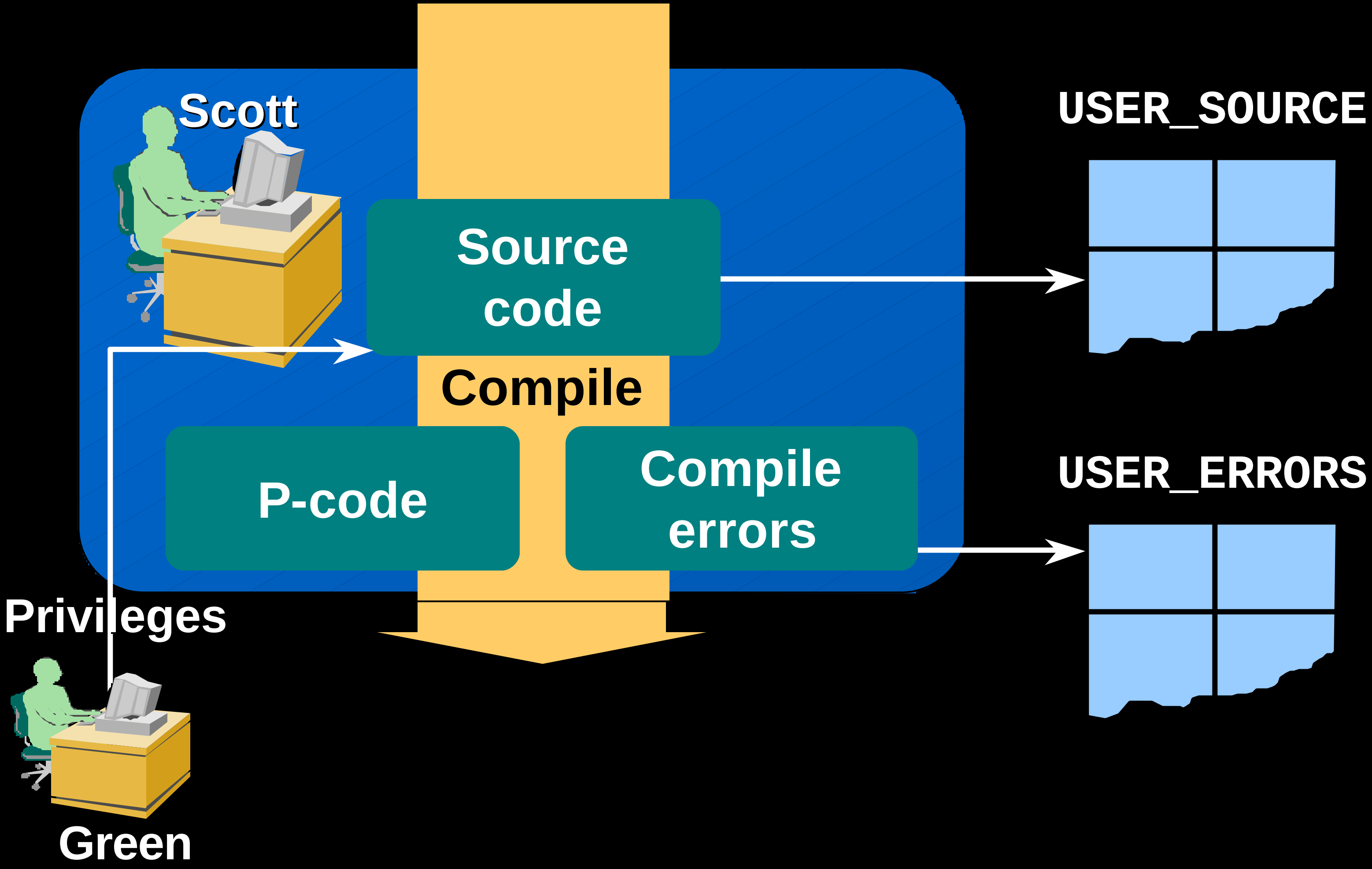
Argument Name	Type	In/Out	Default?
P_NAME	VARCHAR2(30)	IN	DEFAULT
P_LOC	NUMBER(4)	IN	DEFAULT

FUNCTION TAX RETURNS NUMBER

Argument Name	Type	In/Out	Default?
P_VALUE	NUMBER	IN	

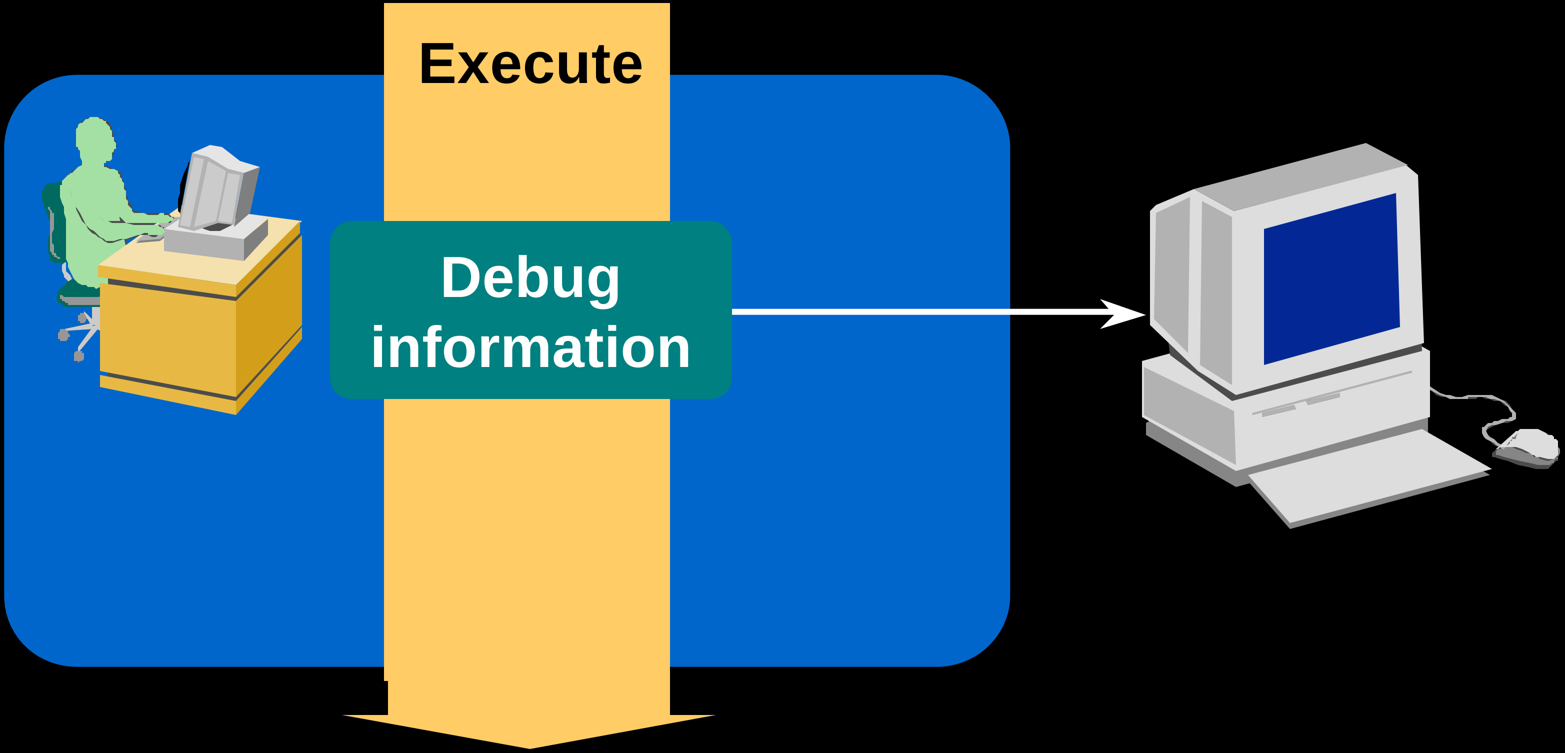
Copyright © Oracle Corporation, 2001. All rights reserved.

Summary



Copyright © Oracle Corporation, 2001. All rights reserved.

Summary



Copyright © Oracle Corporation, 2001. All rights reserved.

Practice 11 Overview

This practice covers the following topics:

- Re-creating the source file for a procedure
- Re-creating the source file for a function

Copyright © Oracle Corporation, 2001. All rights reserved.

12

Creating Packages

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- Describe packages and list their possible components
- Create a package to group together related variables, cursors, constants, exceptions, procedures, and functions
- Designate a package construct as either public or private
- Invoke a package construct
- Describe a use for a bodiless package

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview of Packages

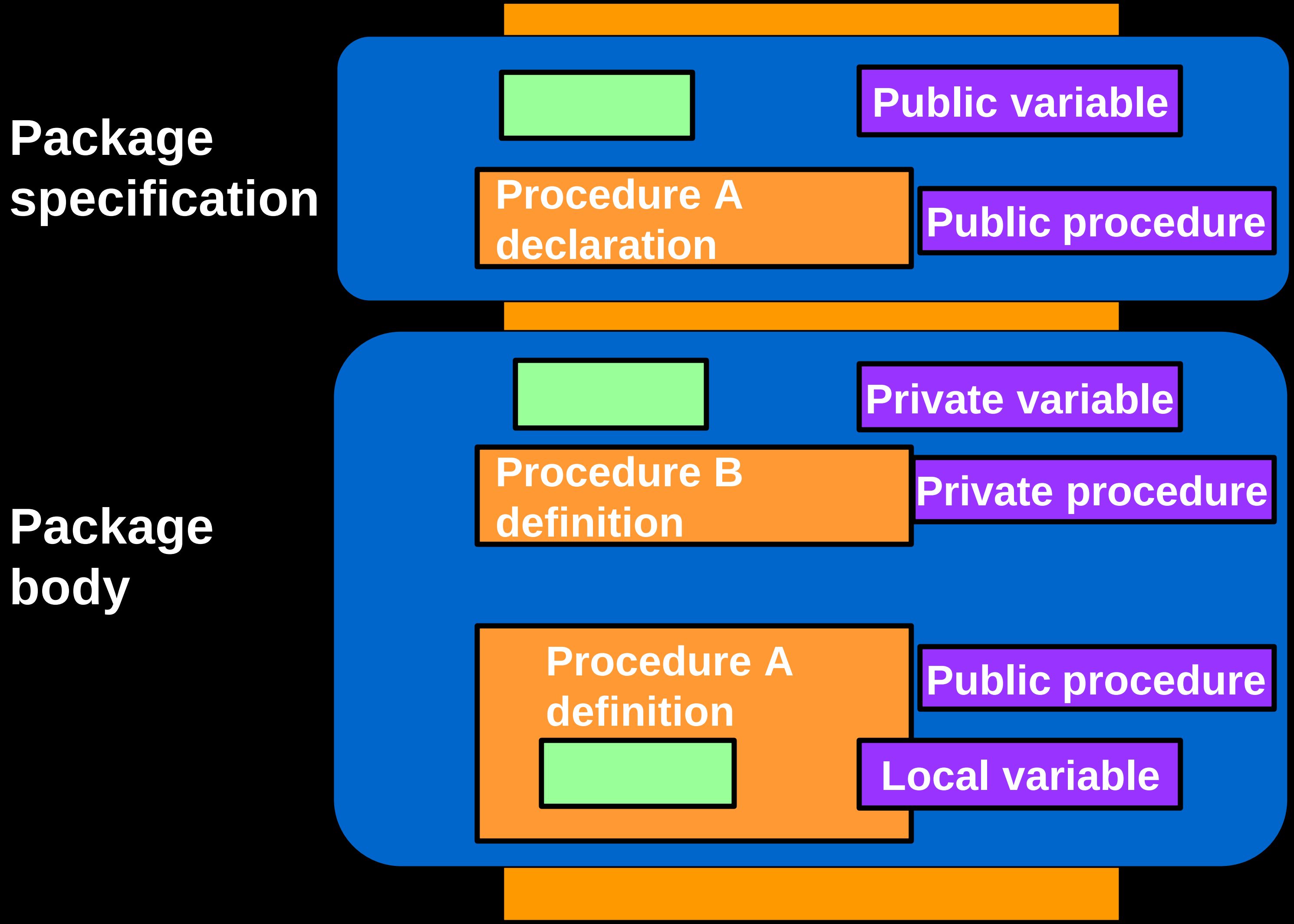
Packages:

- Group logically related PL/SQL types, items, and subprograms
- Consist of two parts:
  - Specification
  - Body
- Cannot be invoked, parameterized, or nested
- Allow the Oracle server to read multiple objects into memory at once

ORACLE

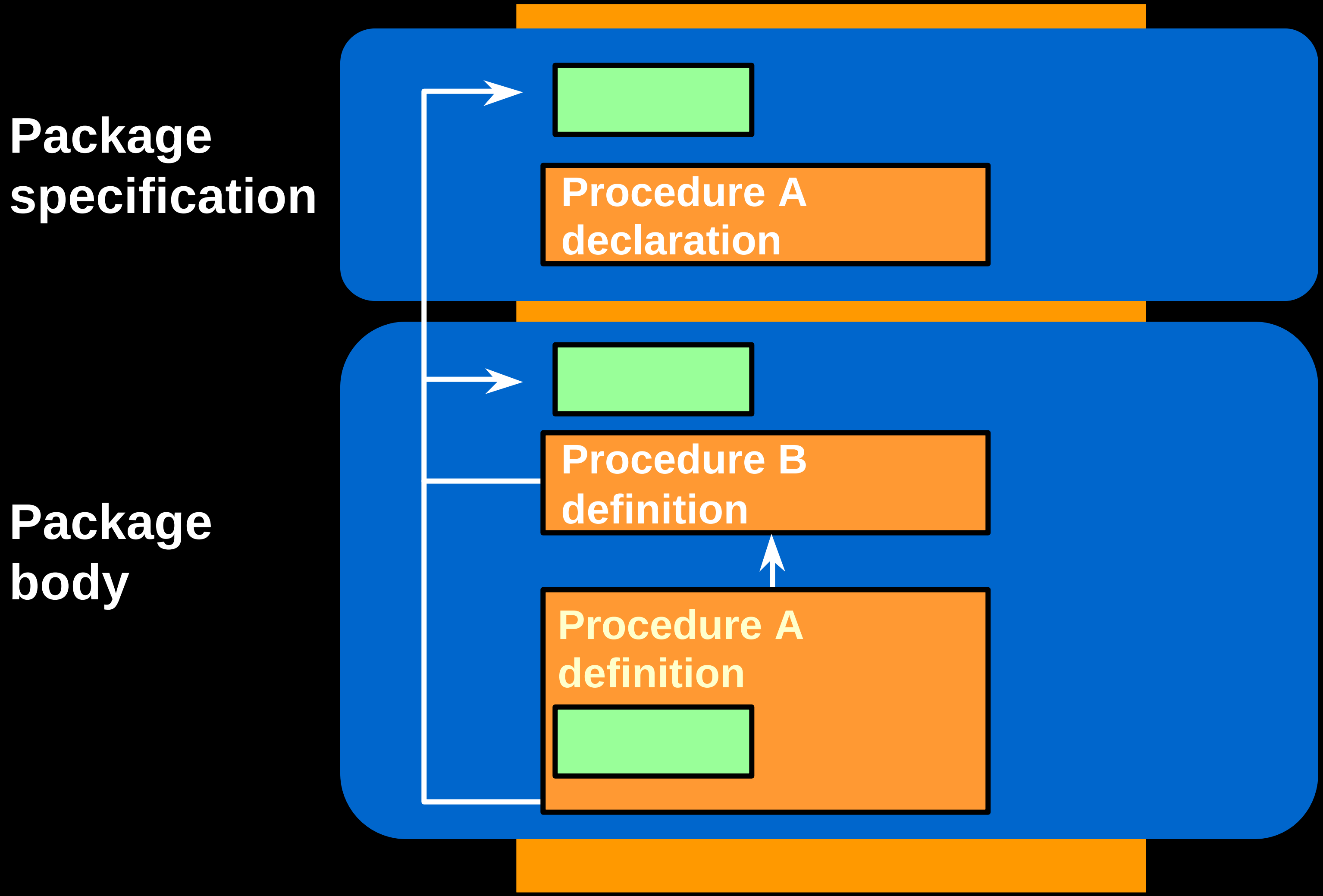
Copyright © Oracle Corporation, 2001. All rights reserved.

Components of a Package



Copyright © Oracle Corporation, 2001. All rights reserved.

Referencing Package Objects



Copyright © Oracle Corporation, 2001. All rights reserved.

Developing a Package

- Saving the text of the `CREATE PACKAGE` statement in two different SQL files facilitates later modifications to the package.
- A package specification can exist without a package body, but a package body cannot exist without a package specification.

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating the Package Specification

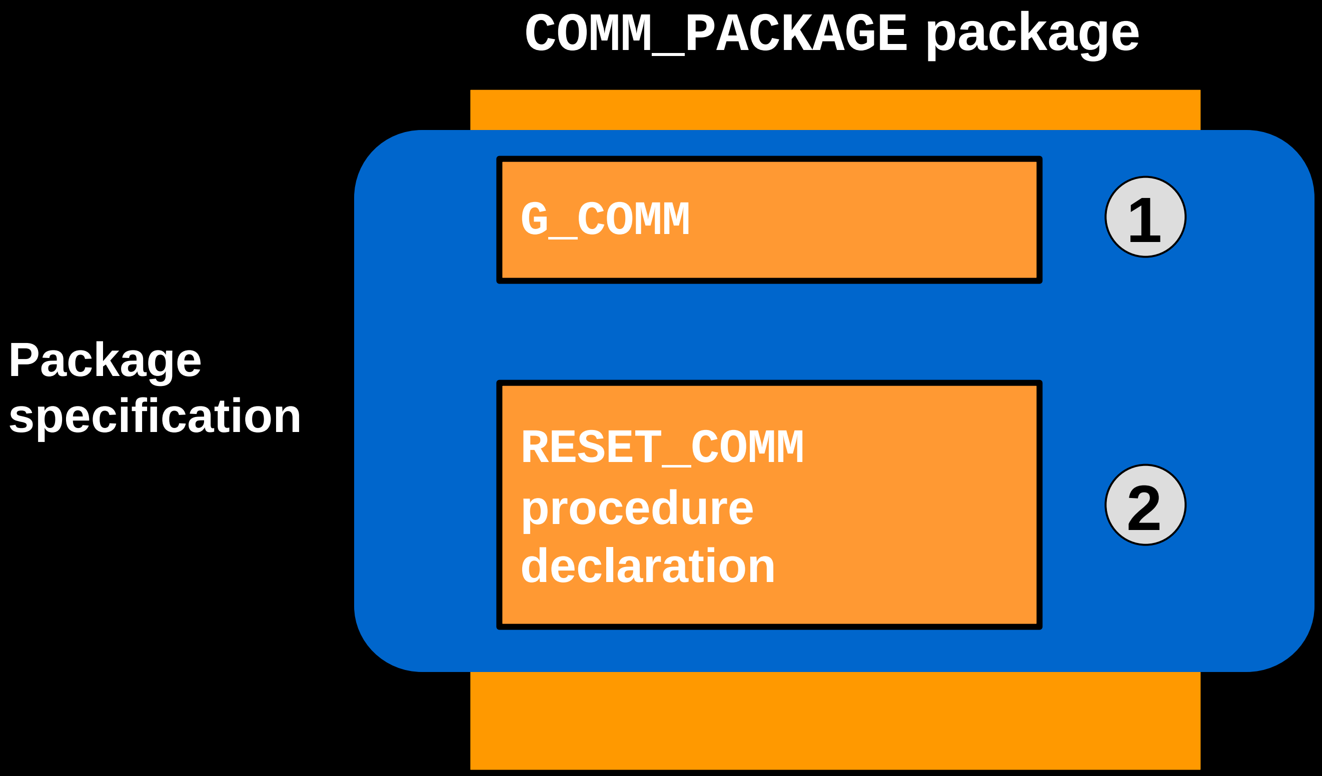
Syntax:

```
CREATE [OR REPLACE] PACKAGE package_name
IS|AS
    public type and item declarations
    subprogram specifications
END package_name;
```

- **The `REPLACE` option drops and recreates the package specification.**
- **Variables declared in the package specification are initialized to `NULL` by default.**
- **All the constructs declared in a package specification are visible to users who are granted privileges on the package.**

Copyright © Oracle Corporation, 2001. All rights reserved.

Declaring Public Constructs



Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Package Specification: Example

```
CREATE OR REPLACE PACKAGE comm_package IS
  g_comm NUMBER := 0.10;  --initialized to 0.10
  PROCEDURE reset_comm
  (p_comm   IN  NUMBER);
END comm_package;
/
```

```
Package created.
```

- G_COMM is a global variable and is initialized to 0.10.
- RESET_COMM is a public procedure that is implemented in the package body.

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating the Package Body

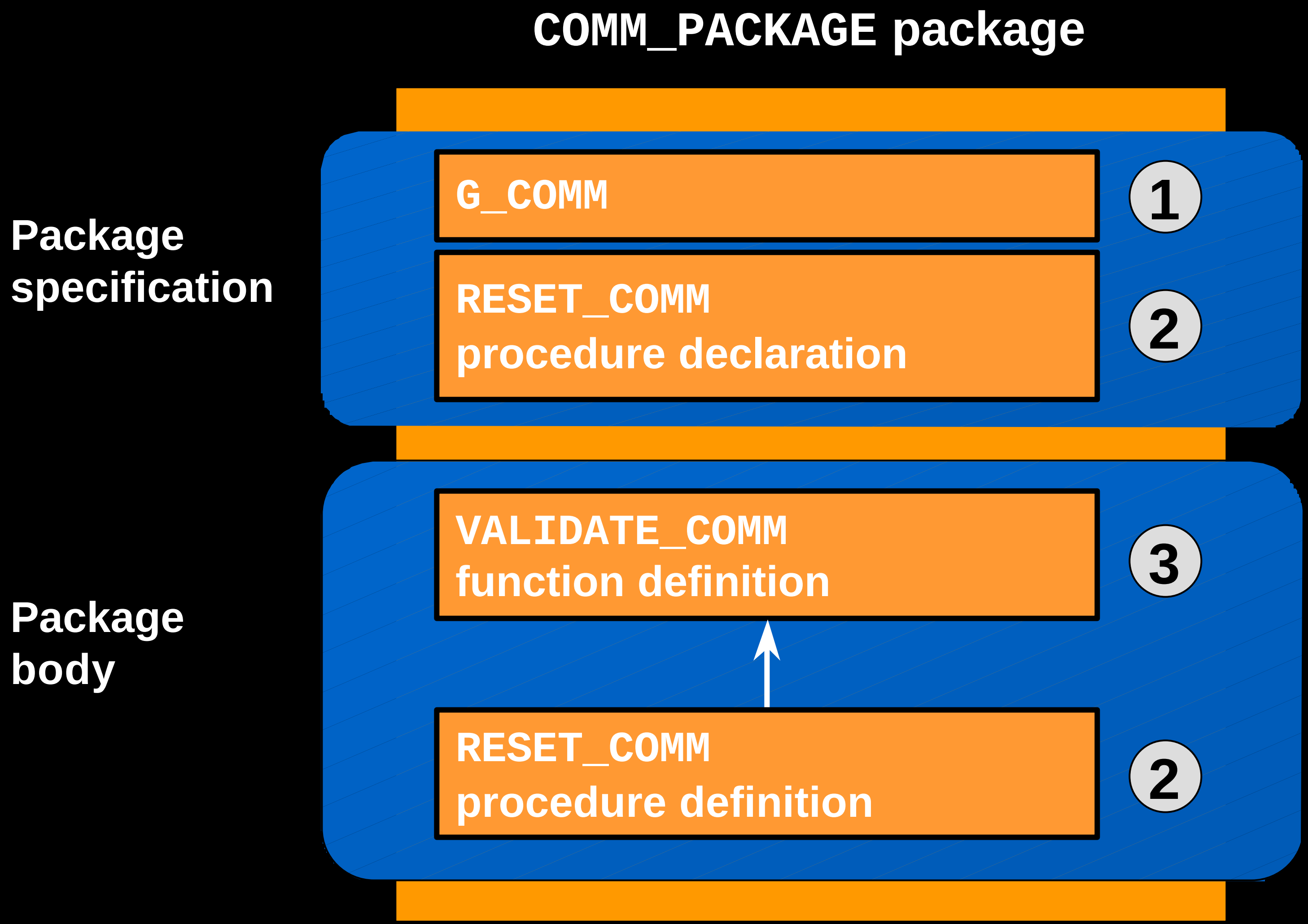
Syntax:

```
CREATE [OR REPLACE] PACKAGE BODY package_name
IS|AS
    private type and item declarations
    subprogram bodies
END package_name;
```

- The REPLACE option drops and recreates the package body.
- Identifiers defined only in the package body are private constructs. These are not visible outside the package body.
- All private constructs must be declared before they are used in the public constructs.

Copyright © Oracle Corporation, 2001. All rights reserved.

Public and Private Constructs


COMM_PACKAGE package
Package specification
G_COMM
1
RESET_COMM procedure declaration
2
Package body
VALIDATE_COMM function definition
3
RESET_COMM procedure definition
2

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Package Body: Example

comm_pack.sql

```
CREATE OR REPLACE PACKAGE BODY comm_package
IS
   FUNCTION  validate_comm (p_comm IN NUMBER)
    RETURN BOOLEAN
   IS
     v_max_comm    NUMBER;
   BEGIN
     SELECT    MAX(commission_pct)
      INTO     v_max_comm
      FROM     employees;
     IF   p_comm > v_max_comm THEN RETURN(FALSE);
     ELSE   RETURN(TRUE);
     END IF;
   END validate_comm;
...
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Package Body: Example

comm_pack.sql

```
  PROCEDURE  reset_comm (p_comm   IN  NUMBER)
  IS
  BEGIN
   IF  validate_comm(p_comm)
    THEN   g_comm:=p_comm;  --reset global variable
   ELSE
    RAISE_APPLICATION_ERROR(-20210,'Invalid commission');
   END IF;
  END reset_comm;
END comm_package;
/
```

```
Package body created.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking Package Constructs

Example 1: Invoke a function from a procedure within the same package.

```
CREATE OR REPLACE PACKAGE BODY comm_package IS
   . . .
 PROCEDURE reset_comm
  (p_comm   IN  NUMBER)
 IS
 BEGIN
  IF validate_comm(p_comm)
  THEN g_comm := p_comm;
  ELSE
    RAISE_APPLICATION_ERROR
         (-20210, 'Invalid commission');
  END IF;
 END reset_comm;
END comm_package;
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking Package Constructs

Example 2: Invoke a package procedure from *i*SQL*Plus.

```
EXECUTE comm_package.reset_comm(0.15)
```

Example 3: Invoke a package procedure in a different schema.

```
EXECUTE scott.comm_package.reset_comm(0.15)
```

Example 4: Invoke a package procedure in a remote database.

```
EXECUTE comm_package.reset_comm@ny(0.15)
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Declaring a Bodiless Package

```
CREATE OR REPLACE PACKAGE global_consts IS
  mile_2_kilo    CONSTANT  NUMBER  :=  1.6093;
  kilo_2_mile    CONSTANT  NUMBER  :=  0.6214;
  yard_2_meter   CONSTANT  NUMBER  :=  0.9144;
  meter_2_yard   CONSTANT  NUMBER  :=  1.0936;
END global_consts;
/

EXECUTE DBMS_OUTPUT.PUT_LINE('20 miles = '||20*
    global_consts.mile_2_kilo||' km')
```

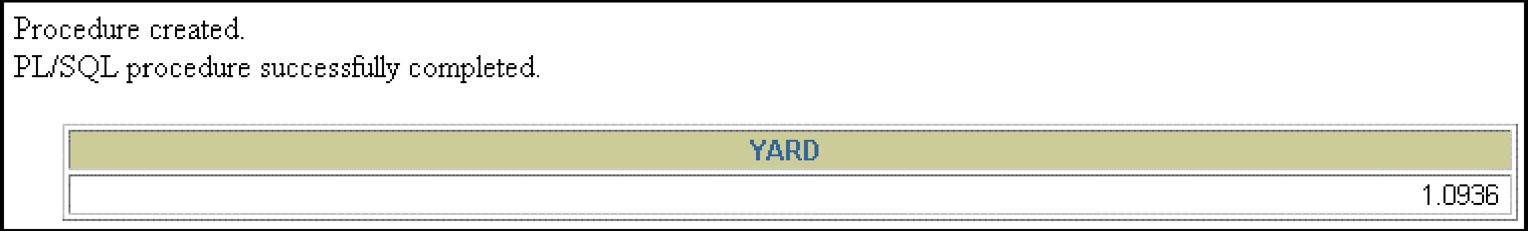
```
Package created.
20 miles = 32.186 km
PL/SQL procedure successfully completed.
```

Copyright © Oracle Corporation, 2001. All rights reserved.

Referencing a Public Variable from a Stand-Alone Procedure

Example:

```
CREATE OR REPLACE PROCEDURE meter_to_yard
            (p_meter IN NUMBER, p_yard OUT NUMBER)
IS
BEGIN
  p_yard := p_meter * global_consts.meter_2_yard;
END meter_to_yard;
/
VARIABLE yard NUMBER
EXECUTE  meter_to_yard (1, :yard)
PRINT yard
```



```
Procedure created.
PL/SQL procedure successfully completed.
```

YARD
1.0936

Copyright © Oracle Corporation, 2001. All rights reserved.

Removing Packages

To remove the package specification and the body, use the following syntax:

```
DROP PACKAGE package_name;
```

To remove the package body, use the following syntax:

```
DROP PACKAGE BODY package_name;
```

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Guidelines for Developing Packages

- Construct packages for general use.
- Define the package specification before the body.
- The package specification should contain only those constructs that you want to be public.
- Place items in the declaration part of the package body when you must maintain them throughout a session or across transactions.
- Changes to the package specification require recompilation of each referencing subprogram.
- The package specification should contain as few constructs as possible.

Copyright © Oracle Corporation, 2001. All rights reserved.

Advantages of Packages

- Modularity: Encapsulate related constructs.
- Easier application design: Code and compile specification and body separately.
- Hiding information:
  - Only the declarations in the package specification are visible and accessible to applications.
  - Private constructs in the package body are hidden and inaccessible.
  - All coding is hidden in the package body.

Copyright © Oracle Corporation, 2001. All rights reserved.

Advantages of Packages

- Added functionality: Persistency of variables and cursors
- Better performance:
  - The entire package is loaded into memory when the package is first referenced.
  - There is only one copy in memory for all users.
  - The dependency hierarchy is simplified.
- Overloading: Multiple subprograms of the same name

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Improve organization, management, security, and performance by using packages
- Group related procedures and functions together in a package
- Change a package body without affecting a package specification
- Grant security access to the entire package

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Hide the source code from users
- Load the entire package into memory on the first call
- Reduce disk access for subsequent calls
- Provide identifiers for the user session

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

Command	Task
`CREATE [OR REPLACE] PACKAGE`	Create (or modify) an existing package specification
`CREATE [OR REPLACE] PACKAGE BODY`	Create (or modify) an existing package body
`DROP PACKAGE`	Remove both the package specification and the package body
`DROP PACKAGE BODY`	Remove the package body only

Copyright © Oracle Corporation, 2001. All rights reserved.