

What Are Procedures?

- Are a type of subprogram that perform an action
- Can be stored in the database as a schema object
- Promote reusability and maintainability



ORACLE

10-1

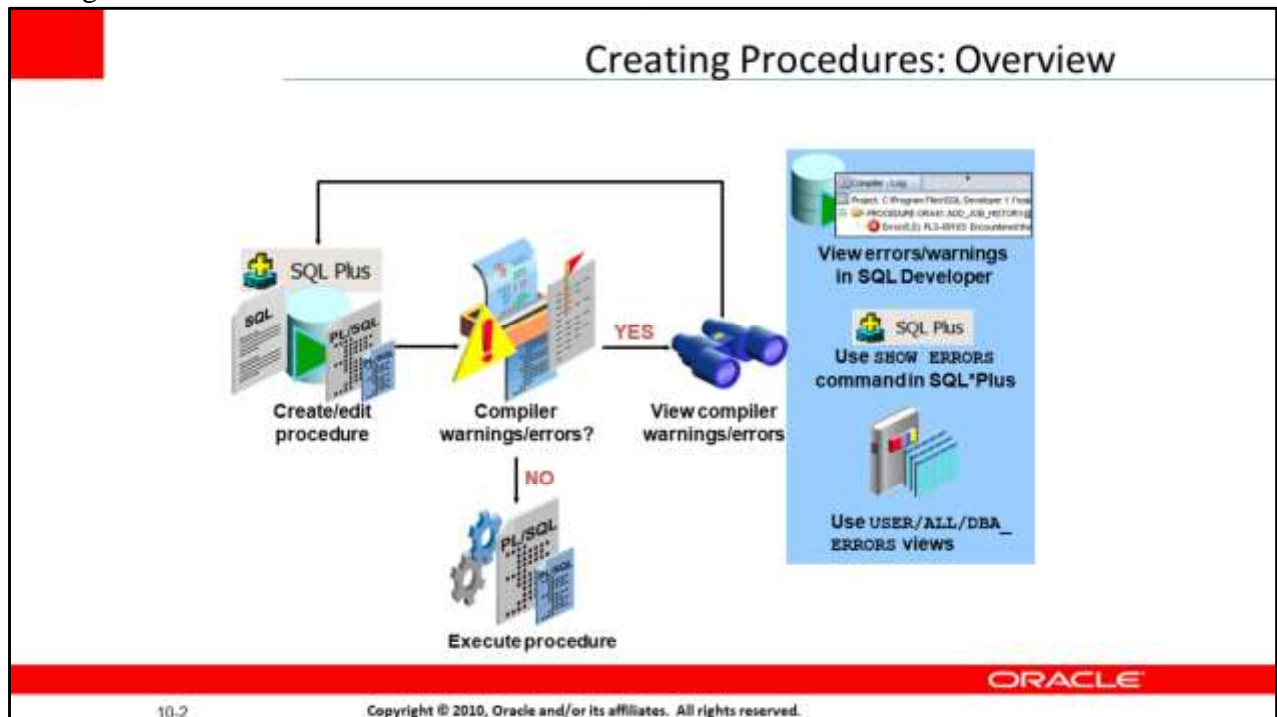
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Definition of a Procedure

A procedure is a named PL/SQL block that can accept parameters (sometimes referred to as arguments). Generally, you use a procedure to perform an action. It has a header, a declaration section, an executable section, and an optional exception-handling section. A procedure is invoked by using the procedure name in the execution section of another PL/SQL block.

A procedure is compiled and stored in the database as a schema object. If you are using the procedures with Oracle Forms and Reports, then they can be compiled within the Oracle Forms or Oracle Reports executables.

Procedures promote reusability and maintainability. When validated, they can be used in any number of applications. If the requirements change, only the procedure needs to be updated.



Creating Procedures: Overview

To develop a procedure using a tool such as SQL Developer, perform the following steps:

1. Create the procedure using SQL Developer's Object Navigator tree or the SQL Worksheet area.
2. Compile the procedure. The procedure is created in the database and gets compiled. The CREATEPROCEDURE statement creates and stores the source code and the compiled m-code in the database. To compile the procedure, right-click the procedure's name in the Object Navigator tree, and then click Compile.
3. If compilation errors exist, then the m-code is not stored and you must edit the source code to make corrections. You cannot invoke a procedure that contains compilation errors. You can view the compilation errors in SQL Developer, SQL*Plus, or the appropriate data dictionary views as shown in the slide.
4. After successful compilation, execute the procedure to perform the desired action. You can run the procedure using SQL Developer or use the EXECUTE command in SQL*Plus.

Note: If compilation errors occur, use a **CREATE OR REPLACE**

Program Units 1 - 2

PROCEDURE statement to overwrite the existing code if you previously used a **CREATE PROCEDURE** statement. Otherwise, drop the procedure first (using **DROP**) and then execute the **CREATE PROCEDURE** statement.

Creating Procedures with the SQL

CREATE OR REPLACE Statement

- Use the CREATE clause to create a stand-alone procedure that is stored in the Oracle database.
- Use the OR REPLACE option to overwrite an existing procedure.

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter1 [mode] datatype1,
parameter2 [mode] datatype2, ...)]
IS|AS
[local_variable_declarations; ...]
BEGIN
-- actions;
END [procedure_name];
```

PL/SQL block

10-3
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating Procedures with the SQL CREATEORREPLACE Statement You can use the CREATEPROCEDURE SQL statement to create stand-alone procedures that are stored in an Oracle database. A procedure is similar to a miniature program: it performs a specific action. You specify the name of the procedure, its parameters, its local variables, and the BEGIN-END block that contains its code and handles any exceptions.

PL/SQL blocks start with BEGIN, optionally preceded by the declaration of local variables. PL/SQL blocks end with either END or ENDprocedure_name.

The REPLACE option indicates that if the procedure exists, it is dropped and replaced with the new version created by the statement. The REPLACE option does not drop any of the privileges associated with the procedure.

Other Syntactic Elements

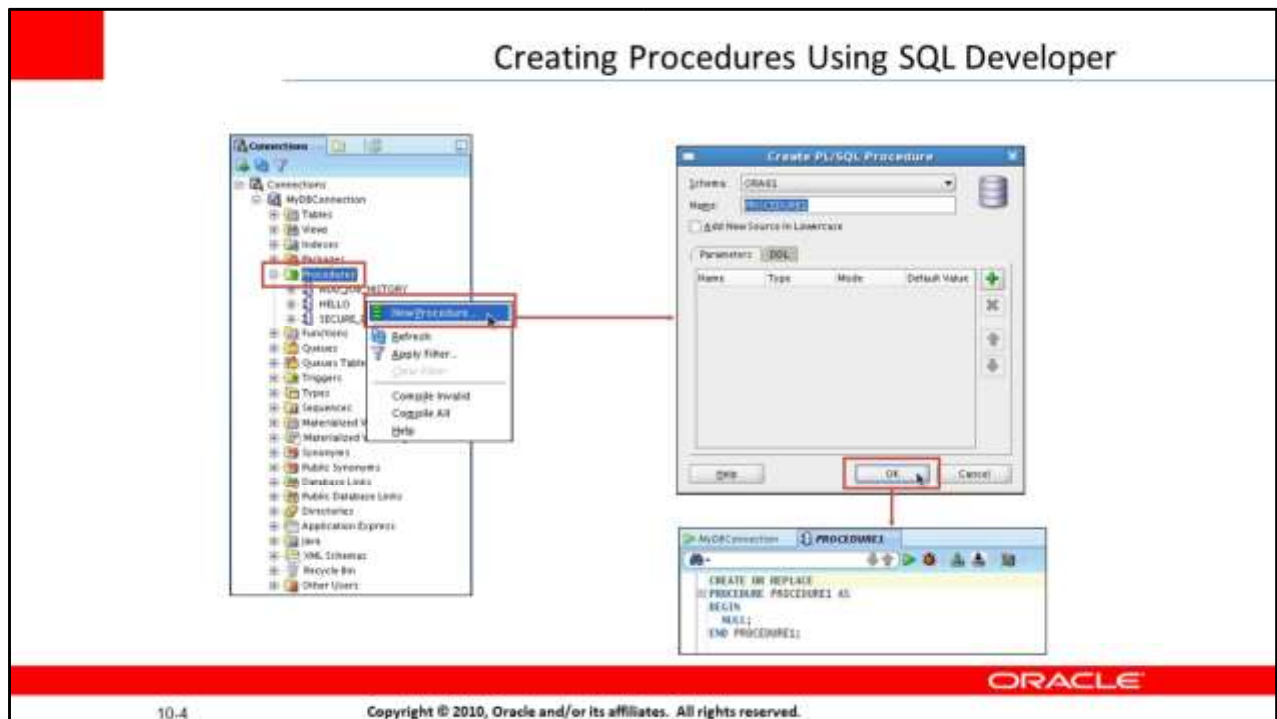
•parameter1 represents the name of a parameter.

The mode option defines how a parameter is used: IN (default), OUT, or INOUT.

•datatype1 specifies the parameter data type, without any precision. Note: Parameters can be considered as local variables. Substitution and host

Program Units 1 - 3

(bind) variables cannot be referenced anywhere in the definition of a PL/SQL stored procedure. The `ORREPLACE` option does not require any change in object security, as long as you own the object and have the `CREATE[ANY] PROCEDURE` privilege.



Creating Procedures Using SQL Developer

1. Right-click the Procedures node on the Connections tabbed page.
2. Select New Procedure from the shortcut menu. The Create PL/SQL Procedure dialog box is displayed. Specify the information for the new procedure, and then click OK to create the subprogram and have it displayed in the Editor window, where you can enter the details.

The components of the Create PL/SQL Procedure dialog box are as follows:

Schema: The database schema in which to create the PL/SQL subprogram

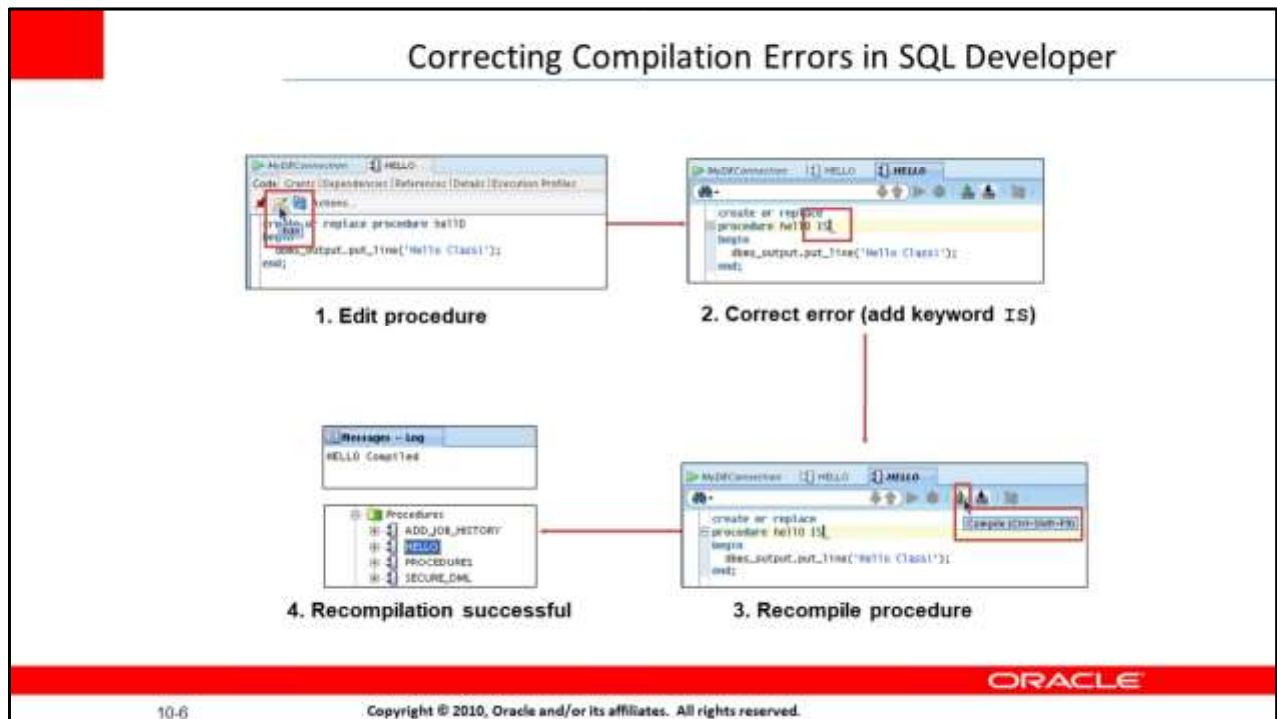
Name: The name of the subprogram that must be unique within a schema

Add New Source in Lowercase: If this option is selected, new text appears in lowercase regardless of the case in which you enter it. This option affects only the appearance of the code, because PL/SQL is not case-sensitive in its execution.

Parameters tab: To add a parameter, click the Add (+) icon. For each

parameter in the procedure to be created, specify the parameter name, data type, mode, and optionally the default Value. Use the Remove (X) icon and the arrow icons to delete and to move a parameter up or down in the list respectively.

DDL tab: This tab contains a read-only display of a SQL statement that reflects the current definition of the subprogram.



Compiling Procedures and Displaying Compilation Errors in SQL Developer You can compile procedures using one of the following two methods: Navigate to the Procedures node in the Object Navigator tree. Rightclick the procedure's name, and then select Compile from the shortcut menu. To view any compilation messages, view the Messages subtab in the Compiler – Log tab.

Edit the procedure using the Edit icon on the procedure's code toolbar. Make the necessary edits, and then click the Compile icon on the code toolbar. To view any compilation messages, view the Messages subtab in the Compiler – Log tab.

Correcting Compilation Errors in SQL Developer

1. Edit the procedure using the Edit icon on the procedure's code toolbar. A new procedure code tab is opened in Read/Write mode.
2. Make the necessary corrections.
3. Click the Compile icon on the code toolbar.
4. To view any compilation messages, view the Messages subtab in the Compiler – Log tab. In addition, if the procedure compiled successfully, the red X on the

What Are Parameters and Parameter Modes?

- Are declared after the subprogram name in the PL/SQL header
- Pass or communicate data between the calling environment and the subprogram
- Are used like local variables but are dependent on their parameter-passing mode:
 - An IN parameter mode (the default) provides values for a subprogram to process
 - An OUT parameter mode returns a value to the caller
 - An IN OUT parameter mode supplies an input value, which may be returned (output) as a modified value

ORACLE

10-8

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

procedure's name in the Object Navigator tree is removed.

Naming Conventions of PL/SQL Structures Used in This Course

The slide table displays some examples of the naming conventions for PL/SQL structures that are used in this course.

What Are Parameters?

Parameters are used to transfer data values to and from the calling environment and the procedure (or subprogram). Parameters are declared in the subprogram header, after the name and before the declaration section for local variables.

Parameters are subject to one of the three parameter-passing modes: IN, OUT, or INOUT.

An IN parameter passes a constant value from the calling environment into the procedure.

An OUT parameter passes a value from the procedure to the calling environment.

An INOUT parameter passes a value from the calling environment to the procedure and a possibly different value from the procedure back to the calling environment using the same parameter.

Formal and Actual Parameters

- Formal parameters: Local variables declared in the parameter list of a subprogram specification
- Actual parameters (or arguments): Literal values, variables, and expressions used in the parameter list of the calling subprogram

```
-- Procedure definition, Formal_parameters
CREATE PROCEDURE raise_sal(p_id NUMBER, p_sal NUMBER) IS
BEGIN
  . . .
END raise_sal;

-- Procedure calling, Actual parameters (arguments)
v_emp_id := 100;
raise_sal(v_emp_id, raise+100);
```

ORACLE

10-9

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

Parameters can be thought of as a special form of local variable, whose input values are initialized by the calling environment when the subprogram is called, and whose output values are returned to the calling environment when the subprogram returns control to the caller.

Formal and Actual Parameters

Formal parameters are local variables that are declared in the parameter list of a subprogram specification. In the first example, in the `raise_sal` procedure, the variable `p_id` and `p_sal` identifiers represent the formal parameters.

The actual parameters can be literal values, variables, and expressions that are provided in the parameter list of a calling subprogram. In the second example, a call is made to `raise_sal`, where the `v_emp_id` variable provides the actual parameter value for the `p_id` formal parameter and 2000 is supplied as the actual parameter value for `p_sal`. Actual parameters:

Are associated with formal parameters during the subprogram call

Can also be expressions, as in the following example:

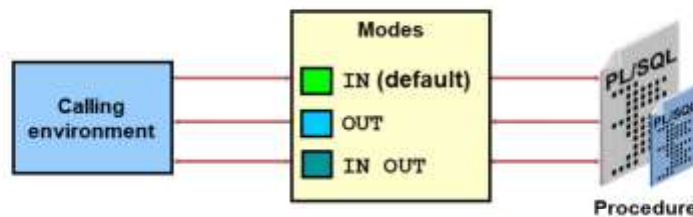
```
raise_sal(v_emp_id, raise+100);
```

The formal and actual parameters should be of compatible data types. If necessary, before assigning the value, PL/SQL converts the data type of the actual parameter value to that of the formal parameter.

Procedural Parameter Modes

- Parameter modes are specified in the formal parameter declaration, after the parameter name and before its data type.
- The `IN` mode is the default if no mode is specified.

```
CREATE PROCEDURE proc_name(param_name [mode] datatype)
```



10-10

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

ORACLE

Note: Actual parameters are also referred to as actual arguments.

Procedural Parameter Modes

When you create a procedure, the formal parameter defines a variable name whose value is used in the executable section of the PL/SQL block. The actual parameter is used when invoking the procedure to provide input values or receive output results.

The parameter mode `IN` is the default passing mode—that is, if no mode is specified with a parameter declaration, the parameter is considered to be an `IN` parameter. The parameter modes `OUT` and `INOUT` must be explicitly specified in their parameter declarations.

The datatype parameter is specified without a size specification. It can be specified:

- As an explicit data type
- Using the `%TYPE` definition
- Using the `%ROWTYPE` definition

Note: One or more formal parameters can be declared, each separated by a comma.

Comparing the Parameter Modes

IN	OUT	IN OUT
Default mode		Must be specified
Value is passed into subprogram	Value is returned to the calling environment	Value passed into subprogram; value 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

ORACLE

10-11

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

Comparing the Parameter Modes

The IN parameter mode is the default mode if no mode is specified in the declaration. The OUT and INOUT parameter modes must be explicitly specified with the parameter declaration.

A formal parameter of IN mode cannot be assigned a value and cannot be modified in the body of the procedure. By default, the IN parameter is passed by reference. An IN parameter can be assigned a default value in the formal parameter declaration, in which case the caller need not provide a value for the parameter if the default applies.

An OUT or INOUT parameter must be assigned a value before returning to the calling environment. The OUT and INOUT parameters cannot be assigned default values. To improve performance with OUT and INOUT parameters, the NOCOPY compiler hint can be used to request to pass by reference.

Note: Using NOCOPY is discussed later in this course.

Using IN Parameters: Example

The example in the slide shows a procedure with two IN parameters. Running the first slide example creates the raise_salary procedure in the database. The second slide example invokes raise_salary and provides the first parameter

Using the OUT Parameter Mode: Example

```
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) IS
BEGIN
  SELECT last_name, salary INTO p_name, p_salary
  FROM   employees
  WHERE  employee_id = p_id;
END query_emp;
/
```

```
SET SERVEROUTPUT ON
DECLARE
  v_emp_name employees.last_name%TYPE;
  v_emp_sal   employees.salary%TYPE;
BEGIN
  query_emp(171, v_emp_name, v_emp_sal);
  DBMS_OUTPUT.PUT_LINE(v_emp_name||' earns '||
    to_char(v_emp_sal, '$999,999.00'));
END;
/
```

ORACLE

10-13

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

value of 176 for the employee ID, and a salary increase of 10 percent for the second parameter value.

To invoke a procedure by using the SQL Worksheet of SQL Developer or by using SQL*Plus, use the following EXECUTE command shown in the second code example in the slide.

To invoke a procedure from another procedure, use a direct call inside an executable section of the calling block. At the location of calling the new procedure, enter the procedure name and actual parameters. For example:

```
...
BEGIN
  raise_salary (176, 10); END;
```

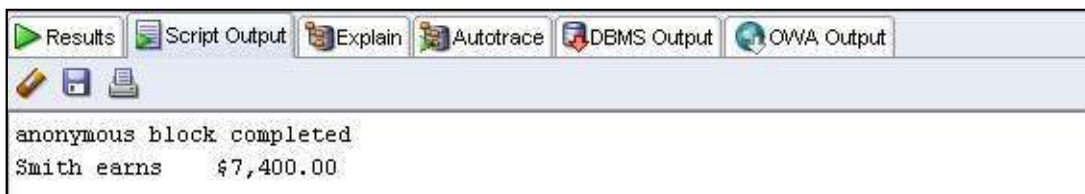
Note: IN parameters are passed as read-only values from the calling environment into the procedure. Attempts to change the value of an IN parameter result in a compile-time error.

Using the OUT Parameters: Example

In the slide example, you create a procedure with OUT parameters to retrieve information about an employee. The procedure accepts the value 171 for employee ID and retrieves the name and salary of the employee with ID 171 into the two OUT parameters. The query_emp procedure has three formal

parameters. Two of them are OUT parameters that return values to the calling environment, shown in the second code box in the slide. The procedure accepts an employee ID value through the p_id parameter. The v_emp_name and v_emp_salary variables are populated with the information retrieved from the query into their two corresponding OUT parameters. The following is the result of running the code in the second code example in the slide.

v_emp_name holds the value Smith and v_emp_salary holds the value 7400:



The screenshot shows a window with a toolbar at the top containing icons for Results, Script Output, Explain, Autotrace, DBMS Output, and OWA Output. Below the toolbar, the text "anonymous block completed" is displayed. Underneath this, a table of results is shown with two columns: the first column contains the text "Smith earns" and the second column contains the value "\$7,400.00".

Column 1	Column 2
Smith earns	\$7,400.00

Note: Make sure that the data type for the actual parameter variables used to retrieve values from the OUT parameters has a size sufficient to hold the data values being returned.

Using the IN OUT Parameter Mode: Example

Calling environment

p_phone_no (before the call)	p_phone_no (after the call)
'8006330575'	'(800) 633-0575'

```
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;  
/
```

```
anonymous block completed  
p_phone_no  
-----  
8006330575  
  
anonymous block completed  
p_phone_no  
-----  
(800) 633-0575
```

ORACLE

10-14

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

Using INOUT Parameters: Example

Using an INOUT parameter, you can pass a value into a procedure that can be updated. The actual parameter value supplied from the calling environment can return either the original unchanged value or a new value that is set within the procedure.

Note: An INOUT parameter acts as an initialized variable.

The slide example creates a procedure with an INOUT parameter to accept a 10-character string containing digits for a phone number. The procedure returns the phone number formatted with parentheses around the first three characters and a hyphen after the sixth digit—for example, the phone string 8006330575 is returned as (800) 633-0575.

The following code uses the b_phone_no host variable of SQL*Plus to provide the input value passed to the FORMAT_PHONE procedure. The procedure is executed and returns an updated string in the b_phone_no host variable. The output of the following code is displayed in the slide above:

```
VARIABLE b_phone_no VARCHAR2(15)  
EXECUTE :b_phone_no := '8006330575'  
PRINT b_phone_no  
EXECUTE format_phone (:b_phone_no)
```

Oracle Database: Develop PL/SQL
Program Units 1 - 14

PRINT b_phone_no

Viewing OUT Parameters:
Using SQL*Plus Host Variables

1. Use SQL*Plus host variables.

2. Execute QUERY_EMP using host variables.

3. Print the host variables.

```
VARIABLE b_name VARCHAR2(25)
VARIABLE b_sal NUMBER
EXECUTE query_emp(171, :b_name, :b_sal)
PRINT b_name b_sal
```

Results | Script Output | Explain | Auditors | DBMS Output | Data Output

anonymous block completed

b_name

SMITH

b_sal

7400

ORACLE

10-16 Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Viewing the OUT Parameters: Using the DBMS_OUTPUT Subroutine

The slide example illustrates how to view the values returned from the OUT parameters in SQL*Plus or the SQL Developer Worksheet.

You can use PL/SQL variables in an anonymous block to retrieve the OUT parameter values. The DBMS_OUTPUT.PUT_LINE procedure is called to print the values held in the PL/SQL variables. The SETSERVEROUTPUT must be ON.

Viewing OUT Parameters: Using SQL*Plus Host Variables

The example in the slide demonstrates how to use SQL*Plus host variables that are created using the VARIABLE command. The SQL*Plus variables are external to the PL/SQL block and are known as host or bind variables. To reference host variables from a PL/SQL block, you must prefix their names with a colon (:). To display the values stored in the host variables, you must use the SQL*Plus PRINT command followed by the name of the SQL*Plus variable (without the colon because this is not a PL/SQL command or context). Note: For details about the VARIABLE command, see the SQL*Plus Command Reference.

Available Notations for Passing Actual Parameters

- When calling a subprogram, you can write the actual parameters using the following notations:
 - Positional: Lists the actual parameters in the same order as the formal parameters
 - Named: Lists the actual parameters in arbitrary order and uses the association operator (`=>`) to associate a named formal parameter with its actual parameter
 - Mixed: Lists some of the actual parameters as positional and some as named
- Prior to Oracle Database 11g, only the positional notation is supported in calls from SQL
- Starting in Oracle Database 11g, named and mixed notation can be used for specifying arguments in calls to PL/SQL subroutines from SQL statements

ORACLE

10-17

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

Oracle Database: Develop PL/SQL

Syntax for Passing Parameters

When calling a subprogram, you can write the actual parameters using the following notations:

Positional: You list the actual parameter values in the same order in which the formal parameters are declared. This notation is compact, but if you specify the parameters (especially literals) in the wrong order, the error can be hard to detect. You must change your code if the procedure's parameter list changes.


Named: You list the actual values in arbitrary order and use the association operator to associate each actual parameter with its formal parameter by name. The PL/SQL association operator is an "equal" sign followed by an "is greater than" sign, without spaces: `=>`. The order of the parameters is not significant. This notation is more verbose, but makes your code easier to read and maintain. You can sometimes avoid changing your code if the procedure's parameter list changes, for example, if the parameters are reordered or a new optional parameter is added.

Mixed: You list the first parameter values by their position and the remainder by using the special syntax of the named method. You can

use this notation to call procedures that have some required parameters, followed by some optional parameters.

Passing Actual Parameters: Creating the add_dept Procedure

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



The screenshot shows the SQL Developer interface with the 'Script Output' tab selected. It displays the message: 'PROCEDURE add_dept: Compiled.'

10-18

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

ORACLE

Passing Parameters: Examples

In the slide example, the add_dept procedure declares two IN formal parameters: p_name and p_loc. The values of these parameters are used in the INSERT statement to set the department_name and location_id columns, respectively.

Passing Actual Parameters: Examples

```
-- Passing parameters using the positional notation.  
EXECUTE add_dept ('TRAINING', 2500)
```



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
200	TRAINING		2500

1 row selected

```
-- Passing parameters using the named notation.  
EXECUTE add_dept (p_loc=>2400, p_name=>'EDUCATION')
```



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
200	EDUCATION		2400

1 row selected

ORACLE

10-19

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

Passing Actual Parameters: Examples

Passing actual parameters by position is shown in the first call to execute `add_dept` in the first code example in the slide. The first actual parameter supplies the value `TRAINING` for the name formal parameter. The second actual parameter value of `2500` is assigned by position to the `loc` formal parameter.

Passing parameters using the named notation is shown in the second code example in the slide. The `loc` actual parameter, which is declared as the second formal parameter, is referenced by name in the call, where it is associated with the actual value of `2400`. The name parameter is associated with the value `EDUCATION`. The order of the actual parameters is irrelevant if all parameter values are specified.

Note: You must provide a value for each parameter unless the formal parameter is assigned a default value. Specifying default values for formal parameters is discussed next.

Using the DEFAULT Option for the Parameters

- Defines default values for parameters
- Provides flexibility by combining the positional and named parameter-passing syntax

```
CREATE OR REPLACE PROCEDURE add_dept(  
  p_name departments.department_name%TYPE := 'Unknown',  
  p_loc  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;
```

```
EXECUTE add_dept  
EXECUTE add_dept ('ADVERTISING', p_loc => 1200)  
EXECUTE add_dept (p_loc => 1200)
```

ORACLE

10-20

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

Using the DEFAULT Option for Parameters

You can assign a default value to an IN parameter as follows:

The assignment operator ($:=$), as shown for the name parameter in the slide

The DEFAULT option, as shown for the p_loc parameter in the slide

When default values are assigned to formal parameters, you can call the procedure without supplying an actual parameter value for the parameter. Thus, you can pass different numbers of actual parameters to a subprogram, either by accepting or by overriding the default values as required. It is recommended that you declare parameters without default values first. Then, you can add formal parameters with default values without having to change every call to the procedure.

Note: You cannot assign default values to the OUT and INOUT parameters. The second code box in the slide shows three ways of invoking the add_dept procedure:

The first example assigns the default values for each parameter. The second example illustrates a combination of position and named notation to assign values. In this case, using named notation is presented as an example.

The last example uses the default value for the name parameter, Unknown, and the supplied value for the p_loc parameter.

Using the DEFAULT Option for Parameters (continued)

The following is the result of the second slide code example in the previous slide:

```
anonymous block completed
anonymous block completed
anonymous block completed
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
280	TRAINING		2500
290	EDUCATION		2400
300	Unknown		1700
310	ADVERTISING		1200
320	Unknown		1200
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700
120	Treasury		1700
130	Corporate Tax		1700
140	Control And Credit		1700
150	Shareholder Services		1700
160	Benefits		1700
170	Manufacturing		1700
180	Construction		1700
190	Contracting		1700
200	Operations		1700
210	IT Support		1700
220	NOC		1700
230	IT Helpdesk		1700
240	Government Sales		1700
250	Retail Sales		1700
260	Recruiting		1700
270	Payroll		1700

32 rows selected

Usually, you can use named notation to override the default values of formal parameters. However, you cannot skip providing an actual parameter if there is no default value provided for a formal parameter.

Note: All the positional parameters should precede the named parameters in a subprogram call. Otherwise, you receive an error message, as shown in the following example: EXECUTE add_dept(p_name=>'new dept', 'new location')

```

Error starting at line 1 in command:
EXECUTE add_dept(p_name=>'new dept', 'new location')
Error report:
ORA-06550: line 1, column 36:
PLS-00312: a positional parameter association may not follow a named association
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored
06550.00000 - "line %s, column %s:\n%s"
*Cause:      Usually a PL/SQL compilation error.
*Action:

```

Calling Procedures

- You can call procedures using anonymous blocks, another procedure, or packages.
- You must own the procedure or have the EXECUTE privilege.

```

CREATE OR REPLACE PROCEDURE process_employees
IS
  CURSOR cur_emp_cursor IS
    SELECT employee_id
    FROM   employees;
BEGIN
  FOR emp_rec IN cur_emp_cursor
  LOOP
    raise_salary(emp_rec.employee_id, 10);
  END LOOP;
  COMMIT;
END process_employees;
/

```

PROCEDURE process_employees compiled.

ORACLE

10-22

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

Calling Procedures

You can invoke procedures by using:

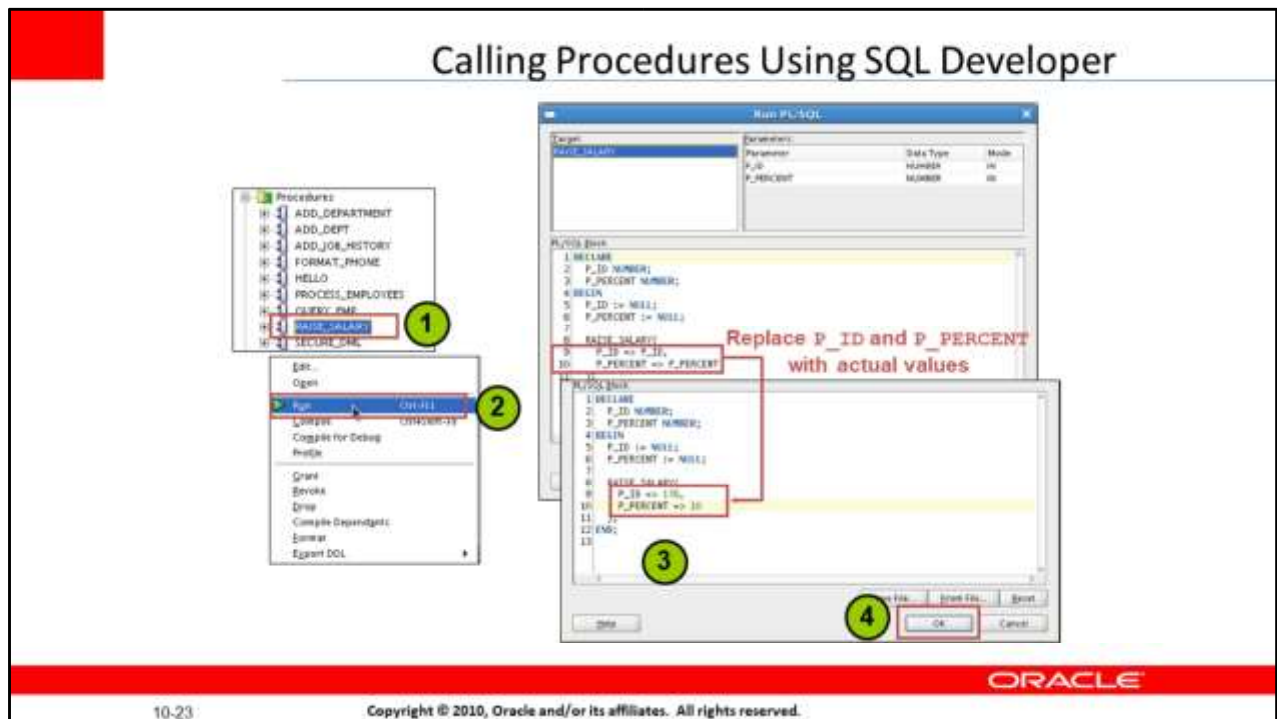
Anonymous blocks

Another procedure or PL/SQL subprogram

Examples on the preceding pages have illustrated how to use anonymous blocks (or the EXECUTE command in SQL Developer or SQL*Plus).

The example in the slide shows you how to invoke a procedure from another stored procedure. The PROCESS_EMPLOYEES stored procedure uses a cursor to process all the records in the EMPLOYEES table and passes each employee's ID to the RAISE_SALARY procedure, which results in a 10% salary increase across the company.

Note: You must own the procedure or have the EXECUTE privilege.



Calling Procedures Using SQL Developer

In the slide example, the raise_salary procedure is called to raise the current salary of employee 176 (\$ 8,600) by 10 percent as follows:

1. Right-click the procedure name in the Procedures node, and then click Run. The Run PL/SQL dialog box is displayed.
2. In the PL/SQL Block section, change the displayed formal IN and IN/OUT parameter specifications displayed after the association operator, “=>” to the actual values that you want to use for running or debugging the function or procedure. For example, to raise the current salary of employee 176 from 8,600 by 10 percent, you can call the raise_salary procedure as shown in the slide. Provide the values for the ID and PERCENT input parameters that are specified as 176 and 10 respectively. This is done by changing the displayed ID => ID with ID => 176 and PERCENT => PERCENT with PERCENT => 10.
3. Click OK. SQL Developer runs the procedure. The updated salary of 9,460 is shown below:

Results		
Script Output Explain Autotrace DBMS Output OWA Output		
Results:		
EMPLOYEE_ID	SALARY	
1	176	9460