ORACLE

Academy

Database Programming with PL/SQL

8-1

Creating Procedures

ORACLE
Academy

Objectives

- This lesson covers the following objectives:
  - Differentiate between anonymous blocks and subprograms
  - Identify the benefits of subprograms
  - Define a stored procedure
  - Create a procedure
  - Describe how a stored procedure is invoked
  - List the development steps for creating a procedure
  - Create a nested subprogram in the declarative section of a procedure

Purpose

- There are times that you want to give a set of steps a name
- For example, if you're told to take notes, you know that this means you need to get out a piece of paper and a pencil and prepare to write
- So far you have learned to write and execute anonymous PL/SQL blocks (blocks that do not have a name associated with them)

Purpose

- Next you will learn how to create, execute, and manage two types of PL/SQL subprograms that are named and stored in the database, resulting in several benefits such as shareability, better security, and faster performance
- Two types of subprograms:
  - Functions
  - Procedures

Differences Between Anonymous Blocks and Subprograms

- As the word “anonymous” indicates, anonymous blocks are unnamed executable PL/SQL blocks
- Because they are unnamed, they can neither be reused nor stored in the database for later use
- While you can store anonymous blocks on your PC, the database is not aware of them, so no one else can share them
- Procedures and functions are PL/SQL blocks that are named, and they are also known as subprograms

ORACLE Academy

Remember, every object stored in the database – tables, views, indexes, synonyms, sequences, and now PL/SQL subprograms – must have a name.

Differences Between Anonymous Blocks and Subprograms

- These subprograms are compiled and stored in the database
- The block structure of the subprograms is similar to the structure of anonymous blocks
- While subprograms can be explicitly shared, the default is to make them private to the owner's schema
- Later subprograms become the building blocks of packages and triggers

Differences Between Anonymous Blocks and Subprograms

- Anonymous blocks

```
DECLARE    (Optional)
   Variables, cursors, etc.;
BEGIN      (Mandatory)
   SQL and PL/SQL statements;
EXCEPTION (Optional)
   WHEN exception-handling actions;
END; (Mandatory)
```

- Subprograms (procedures)

```
CREATE [OR REPLACE] PROCEDURE name [parameters] IS|AS (Mandatory)
  Variables, cursors, etc.; (Optional)
BEGIN      (Mandatory)
   SQL and PL/SQL statements;
EXCEPTION (Optional)
   WHEN exception-handling actions;
END [name]; (Mandatory)
```

ORACLE Academy

Differences Between Anonymous Blocks and Subprograms

- The alternative to an anonymous block is a named block. How the block is named depends on what you are creating
- You can create :
  - a named procedure (does not return values except as out parameters)
  - a function (must return a single value not including out parameters)
  - a package (groups functions and procedures together)
  - a trigger

ORACLE Academy

Differences Between Anonymous Blocks and Subprograms

- The keyword DECLARE is replaced by CREATE PROCEDURE procedure-name IS | AS
- In anonymous blocks, DECLARE states, "this is the start of a block"
- Because CREATE PROCEDURE states, "this is the start of a subprogram," we do not need (and must not use) DECLARE

Differences Between Anonymous Blocks and Subprograms

Anonymous Blocks	Subprograms
Unnamed PL/SQL blocks	Named PL/SQL blocks
Compiled on every execution	Compiled only once, when created
Not stored in the database	Stored in the database
Cannot be invoked by other applications	They are named and therefore can be invoked by other applications
Do not return values	Subprograms called functions must return values
Cannot take parameters	Can take parameters

ORACLE Academy

Almost every programming language supports subprograms. They are a part of structured programming. We can consider it as a separate block of statements (with its own declarations and programming statements), but still under the control of the main program.

The main block calls the subprogram by its name to execute its set of statements. This may be a part of the conditions as well. That means the main program may or may not call subprogram based on certain conditions.

Repeatable tasks are generally separated as subprograms to allow the user to use them as many times as possible. This also improves the readability of the main program, which is being divided into several meaningful tasks, where each task (subprogram) may have its own set of programming statements (including local declarations).

Differences Between Anonymous Blocks and Subprograms

- The keyword DECLARE is replaced by CREATE PROCEDURE procedure-name IS | AS
- In anonymous blocks, DECLARE states, "this is the start of a block"
- Because CREATE PROCEDURE states, "this is the start of a subprogram," we do not need (and must not use) DECLARE

Benefits of Subprograms

- Procedures and functions have many benefits due to the modularizing of the code:
  - Easy maintenance: Modifications need only be done once to improve multiple applications and minimize testing
  - Code reuse: Subprograms are located in one place
- When compiled and validated, they can be used and reused in any number of applications

Benefits of Subprograms

- Improved data security: Indirect access to database objects is permitted by the granting of security privileges on the subprograms
- By default, subprograms run with the privileges of the subprogram owner, not the privileges of the user
- Data integrity: Related actions can be grouped into a block and are performed together ("Statement Processed") or not at all

ORACLE Academy

Subprogram security and privileges will be covered in a later section. For now, if a subprogram contains a SQL statement which SELECTs from a table, the subprogram owner needs SELECT privileges on that table, but the user does not. This means that the only way the user can access the table is through the subprogram, not by any other route. Therefore the user cannot see sensitive or confidential data unless the subprogram code explicitly allows it.

Benefits of Subprograms

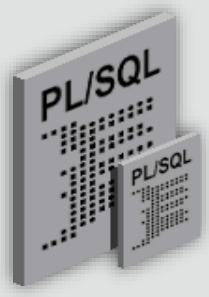
- Improved performance: You can reuse compiled PL/SQL code that is stored in the shared SQL area cache of the server
- Subsequent calls to the subprogram avoid compiling the code again
- Also, many users can share a single copy of the subprogram code in memory
- Improved code clarity: By using appropriate names and conventions to describe the action of the routines, you can reduce the need for comments, and enhance the clarity of the code

ORACLE Academy

Improved performance: a subprogram is compiled only once, when it is (re)CREATEd. An anonymous block is compiled (while the user waits) every time it is executed.

Procedures and Functions : Similarities

- Are named PL/SQL blocks
- Are called PL/SQL subprograms
- Have block structures similar to anonymous blocks:
  - Optional parameters
  - Optional declarative section (but the DECLARE keyword changes to IS or AS)
  - Mandatory executable section
  - Optional section to handle exceptions
- Procedures and functions can both return data as OUT and IN OUT parameters



ORACLE Academy

Packages will be covered in a later section. For now, a package is a group of several related procedures and/or functions which are created and managed as a single unit. Packages are very powerful and important, but the "building blocks" are still procedures and functions.

Procedures and Functions : Differences

- A function MUST return a value using the RETURN statement
- A procedure can only return a value using an OUT or an IN OUT parameter
- The return statement in a function returns control to the calling program and returns the results of the function
- The return statement within a procedure is optional
- It returns control to the calling program before all of the procedure's code has been executed
- Functions can be called from SQL, procedures cannot
- Functions are considered expressions, procedures are not

What Is a Procedure?

- A procedure is a named PL/SQL block that can accept parameters
- Generally, you use a procedure to perform an action (sometimes called a “side-effect”)
- A procedure is compiled and stored in the database as a schema object
  - Shows up in USER_OBJECTS as an object type of PROCEDURE
  - More details in USER_PROCEDURES
  - Detailed PL/SQL code in USER_SOURCE

ORACLE Academy

Syntax for Creating Procedures

- Parameters are optional
- Mode defaults to IN
- Datatype can be either explicit (for example, VARCHAR2) or implicit with %TYPE
- Body is the same as an anonymous block

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

ORACLE Academy

PLSQL 8-1
Creating Procedures

There is no difference between IS and AS, either can be used.

An option of the CREATE PROCEDURE statement is to follow CREATE by OR REPLACE. The advantage of doing so is that should you have already created the procedure in the database, you will not get an error. On the other hand, should the previous definition be a different procedure of the same name, you will not be warned, and the old procedure will be lost.

There can be any number of parameters, each followed by a mode and a type. The modes are IN (read-only), OUT (write-only), and IN OUT (read and write). You will learn more about parameters and their modes in the next two lessons.

Syntax for Creating Procedures

- Use CREATE PROCEDURE followed by the name, optional parameters, and keyword IS or AS
- Add the OR REPLACE option to overwrite an existing procedure
- Write a PL/SQL block containing local variables, a BEGIN, and an END (or END procedure_name)

```
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

ORACLE Academy

Procedure: Example

- In the following example, the add_dept procedure inserts a new department with the department_id 280 and department_name ST-Curriculum
- The procedure declares two variables, v_dept_id and v_dept_name, in the declarative section

```
CREATE OR REPLACE PROCEDURE add_dept IS
  v_dept_id    dept.department_id%TYPE;
  v_dept_name  dept.department_name%TYPE;
BEGIN
  v_dept_id   := 280;
  v_dept_name := 'ST-Curriculum';
  INSERT INTO dept(department_id, department_name)
    VALUES(v_dept_id, v_dept_name);
  DBMS_OUTPUT.PUT_LINE('Inserted '|| SQL%ROWCOUNT || ' row.');
END;
```

- The declarative section of a procedure starts immediately after the procedure declaration and does not begin with the keyword DECLARE
- This procedure uses the SQL%ROWCOUNT cursor attribute to check if the row was successfully inserted
- SQL%ROWCOUNT should return 1 in this case

ORACLE Academy

In a real-life procedure, the v_dept_id and v_dept_name values would be passed into the procedure as parameters, not hard-coded as shown here.

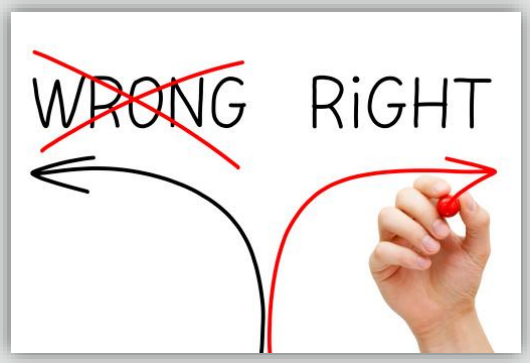
Procedure: Example

```
CREATE OR REPLACE PROCEDURE add_dept IS
  v_dept_id    dept.department_id%TYPE;
  v_dept_name  dept.department_name%TYPE;
BEGIN
  v_dept_id   := 280;
  v_dept_name := 'ST-Curriculum';
  INSERT INTO dept(department_id, department_name)
    VALUES(v_dept_id, v_dept_name);
  DBMS_OUTPUT.PUT_LINE('Inserted '|| SQL%ROWCOUNT || '
row.');
END;
```

ORACLE Academy

Invoking Procedures

- You can invoke (execute) a procedure from:
  - An anonymous block
  - Another procedure
  - A calling application
- Note: You cannot invoke a procedure from inside a SQL statement such as SELECT



ORACLE Academy

Invoking the Procedure from Application Express

- To invoke (execute) a procedure in Oracle Application Express, write and run a small anonymous block that invokes the procedure
- For example:

```
BEGIN
  add_dept;
END;

SELECT department_id, department_name FROM dept WHERE
department_id=280;
```

- The select statement at the end confirms that the row was successfully inserted

ORACLE Academy

Correcting Errors in CREATE PROCEDURE Statements

- If compilation errors exist, Application Express displays them in the output portion of the SQL Commands window
- You must edit the source code to make corrections
- When a subprogram is CREATEd, the source code is stored in the database even if compilation errors occurred

ORACLE Academy

Correcting Errors in CREATE PROCEDURE Statements

- After you have corrected the error in the code, you need to recreate the procedure
- There are two ways to do this:
  - Use a CREATE OR REPLACE PROCEDURE statement to overwrite the existing code (most common)
  - DROP the procedure first and then execute the CREATE PROCEDURE statement (less common)

Saving Your Work

- Once a procedure has been created successfully, you should save its definition in case you need to modify the code later

```
CREATE OR REPLACE PROCEDURE add_dept IS
   v_dept_id    dept.department_id%TYPE;
   v_dept_name  dept.department_name%TYPE;
BEGIN
   v_dept_id :=280;
   v_dept_name := 'ST-Curriculum';
   INSERT INTO dept(department_id, department_name)
      VALUES(v_dept_id, v_dept_name);
   DBMS_OUTPUT.PUT_LINE('Inserted '|| SQL%ROWCOUNT || ' row');
END;
```

Results | Explain | Describe | Saved SQL | History

```
Procedure created.
```

ORACLE Academy

Saving Your Work

- In the Application Express SQL Commands window, click the SAVE button, then enter a name and optional description for your code
- You can view and reload your code later by clicking on the Saved SQL button in the SQL Commands window

Local Subprograms

- When one procedure invokes another procedure, we would normally create them separately, but we can create them together as a single procedure if we like

```
CREATE OR REPLACE PROCEDURE subproc
  ...
END subproc;

CREATE OR REPLACE PROCEDURE mainproc
  ...
IS BEGIN
  ...
    subproc(...);
  ...
END mainproc;
```

Local Subprograms

- All the code is now in one place, and is easier to read and maintain
- The nested subprogram's scope is limited to the procedure within which it is defined; SUBPROC can be invoked from MAINPROC, but from nowhere else

```
CREATE OR REPLACE PROCEDURE mainproc
  ...
IS
  PROCEDURE subproc (...) IS BEGIN
    ...
  END subproc;
BEGIN
  ...
    subproc(...);
  ...
END mainproc;
```

ORACLE Academy

PLSQL 8-1
Creating Procedures

This is similar to the idea of nested anonymous blocks covered earlier in the course. The example shown here shows nested procedures, but PL/SQL functions can also be nested in this way.

Why don't we just write all the code in a single (normal) procedure?

Answer: to restrict the scope of variables. Normal scoping rules apply, so a variable declared within the nested subprogram cannot be referenced in the outer program. This helps to make it clear which variables are used in which part of the code.

Local Subprograms

- Every time an employee is deleted, we need to insert a row into a logging table
- The nested procedure LOG_EMP is called a Local Subprogram

Local Subprograms

```
CREATE OR REPLACE PROCEDURE delete_emp
  (p_emp_id IN employees.employee_id%TYPE)
IS
  PROCEDURE log_emp (p_emp IN employees.employee_id%TYPE)
  IS BEGIN
    INSERT INTO logging_table VALUES(p_emp, ...);
  END log_emp;
BEGIN
  DELETE FROM employees
    WHERE employee_id = p_emp_id;
  log_emp(p_emp_id);
END delete_emp;
```

ORACLE Academy

Alternative Tools for Developing Procedures

- If you end up writing PL/SQL procedures for a living, there are other free tools that can make this process easier
- For instance, Oracle tools, such as SQL Developer and JDeveloper assist you by:
  - Color-coding commands vs variables vs constants
  - Highlighting matched and mismatched (parentheses)
  - Displaying errors more graphically

Alternative Tools for Developing Procedures

- Enhancing code with standard indentations and capitalization
- Completing commands when typing
- Completing column names from tables

Alternative Tools for Developing Procedures

- To develop a stored procedure when not using Oracle Application Express, perform the following steps:
  - 1. Write the code to create a procedure in an editor or a word processor, and then save it as a SQL script file (typically with a .sql extension)
  - 2. Load the code into one of the development tools such as iSQL*Plus or SQL Developer
  - 3. Create the procedure in the database
  - The CREATE PROCEDURE statement compiles and stores source code and the compiled m-code in the database
  - If compilation errors exist, then the m-code is not stored and you must edit the source code to make corrections

Alternative Tools for Developing Procedures

- To develop a stored procedure when not using Oracle Application Express, perform the following steps:
  - 4. After successful compilation, execute the procedure to perform the desired action
  - Use the EXECUTE command from iSQL*Plus or an anonymous PL/SQL block from environments that support PL/SQL

ORACLE Academy

Terminology

- Key terms used in this lesson included:
  - Anonymous blocks
  - IS or AS
  - Procedures
  - Subprograms

ORACLE Academy

- Anonymous Blocks – Unnamed executable PL/SQL blocks that can not be reused no stored for later use.
- IS or AS – Indicates the DECLARE section of a subprogram.
- Procedures – Named PL/SQL blocks that can accept parameters and are compiled and stored in the database.
- Subprograms – Named PL/SQL blocks that are compiled and stored in the database.

Summary

- In this lesson, you should have learned how to:
  - Differentiate between anonymous blocks and subprograms
  - Identify the benefits of subprograms
  - Define a stored procedure
  - Create a procedure
  - Describe how a stored procedure is invoked
  - List the development steps for creating a procedure
  - Create a nested subprogram in the declarative section of a procedure

ORACLE Academy

ORACLE

Academy

ORACLE

Academy

Database Programming with PL/SQL

8-2

Using Parameters in Procedures

ORACLE
Academy

Objectives

- This lesson covers the following objectives:
  - Describe how parameters contribute to a procedure
  - Define a parameter
  - Create a procedure using a parameter
  - Invoke a procedure that has parameters
  - Differentiate between formal and actual parameters

ORACLE Academy

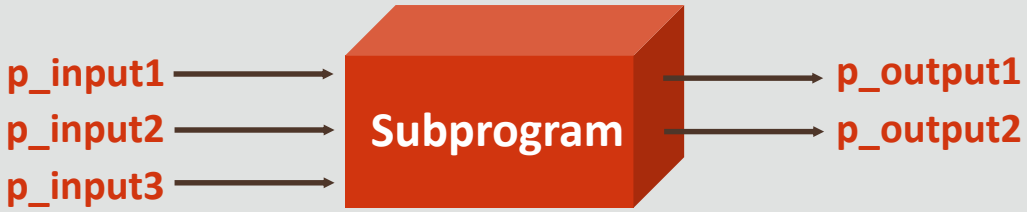
Purpose

- Much time can be spent creating a procedure
- It is important to create the procedure in a flexible way so that it can be used, potentially, for more than one purpose or more than one piece of data
- To make procedures more flexible, it is important that varying data is either calculated or passed into a procedure by using input parameters
- Calculated results can be returned to the caller of a procedure by using parameters

What are Parameters?

- Parameters pass or communicate data between the caller and the subprogram
- You can think of parameters as a special form of a 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
- By convention, parameters are often named with a "p_" prefix

What are Parameters?



ORACLE Academy

What Are Parameters?

- Consider the following example where a math teacher needs to change a student's grade from a C to a B in the student administration system

Student id is 1023	1023
The math class id is 543	543
The new grade is B	B

Calling environment

- In this example, the calling system is passing values for student id, class id, and grade to a subprogram
- Do you need to know the old (before) value for the grade?
- Why or why not?

ORACLE Academy

Answer: normally you would not need to know the old (before) value. The next slide shows the procedure code.

What Are Parameters?

- The change_grade procedure accepts three parameters: p_student_id, p_class_id, and p_grade
- These parameters act like local variables in the change_grade procedure

Student id is 1023	1023
The math class id is 543	543
The new grade is B	B

Calling environment

What Are Parameters?

```
PROCEDURE change_grade (p_student_id IN NUMBER,
p_class_id IN NUMBER, p_grade IN VARCHAR2) IS
BEGIN
…
  UPDATE grade_table
    SET grade = p_grade
    WHERE student_id = p_student_id AND class_id =
p_class_id;
…
END;
```

ORACLE Academy

What Are Arguments?

- Parameters are commonly referred to as arguments
- However, arguments are more appropriately thought of as the actual values assigned to the parameter variables when the subprogram is called at runtime

Student id is 1023	1023
The math class id is 543	543
The new grade is B	B

ORACLE Academy

You were introduced to CURSORs with parameters earlier in the course, and the parameters for procedures and functions are no different.

Parameter means the name, argument means the value. In the example on the previous slide, p_student_id is a parameter while 1023 is an argument.

What Are Arguments?

- Even though parameters are a kind of variable, IN parameters are treated as constants within the subprogram and cannot be changed by the subprogram
- In the previous example, 1023 is an argument passed in to the p_student_id parameter

Student id is 1023	**1023**
The math class id is 543	**543**
The new grade is B	**B**

ORACLE Academy

Creating Procedures with Parameters

- The example shows a procedure with two parameters
- Running this first statement creates the raise_salary procedure in the database
- The second example executes the procedure, passing the arguments 176 and 10 to the two parameters

ORACLE Academy

Creating Procedures with Parameters

```
CREATE OR REPLACE PROCEDURE raise_salary
  (p_id IN my_employees.employee_id%TYPE,
   p_percent IN NUMBER)
IS
BEGIN
  UPDATE my_employees
    SET    salary = salary * (1 + p_percent/100)
    WHERE  employee_id = p_id;
END raise_salary;
```

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

ORACLE Academy

Invoking Procedures with Parameters

- To invoke a procedure from Oracle Application Express, create an anonymous block and use a direct call inside the executable section of the block
- Where you want to call the new procedure, enter the procedure name and parameter values (arguments)
- For example:

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

- You must enter the arguments in the same order as they are declared in the procedure

ORACLE Academy

Invoking Procedures with Parameters

- To invoke a procedure from another procedure, use a direct call inside an executable section of the block
- At the location of calling the new procedure, enter the procedure name and parameter arguments

```
CREATE OR REPLACE PROCEDURE process_employees
IS
  CURSOR emp_cursor IS
    SELECT employee_id
 FROM   my_employees;
BEGIN
   FOR v_emp_rec IN emp_cursor
   LOOP
     raise_salary(v_emp_rec.employee_id, 10);
   END LOOP;
END process_employees;
```

ORACLE Academy

PLSQL 8-2
Using Parameters in Procedures

15

In this example, the PROCESS_EMPLOYEES procedure uses a cursor to process all the records in the EMPLOYEES table and passes each employee's ID to the RAISE_SALARY procedure. If there are 20 rows in the EMPLOYEES table, RAISE_SALARY will execute 20 times. Every employee in turn receives a 10% salary increase.

Types of Parameters

- There are two types of parameters: Formal and Actual
- A parameter-name declared in the procedure heading is called a formal parameter
- The corresponding parameter-name (or value) in the calling environment is called an actual parameter

ORACLE Academy

Types of Parameters

- In the following example, can you identify which parameter is the formal parameter and which parameter is the actual parameter?

```
CREATE OR REPLACE PROCEDURE fetch_emp
  (p_emp_id IN employees.employee_id%TYPE) IS
  ...
END;

/* Now call the procedure from an anonymous block or
subprogram */

BEGIN
  ...
  fetch_emp(v_emp_id);
  ...
END;
```

ORACLE Academy

PLSQL 8-2
Using Parameters in Procedures

Answer: The formal parameter is called p_emp_id and the actual parameter is called v_emp_id. The value stored in v_emp_id in the calling block is passed to the procedure. Within the fetch_emp procedure, that value is referred to as p_emp_id, and the value stored in p_emp_id cannot be changed within fetch_emp.

p_emp_id is the formal parameter and v_emp_id is the actual parameter. We make the distinction because the names can be different (as in this example). However, the data types must be compatible.

Formal Parameters

- Formal parameters are variables that are declared in the parameter list of a subprogram specification
- In the following example, in the procedure raise_sal, the identifiers p_id and p_sal represent formal parameters

```
CREATE PROCEDURE raise_sal(p_id IN NUMBER, p_sal IN
  NUMBER) IS
BEGIN
  ...
END raise_sal;
```

- Notice that the formal parameter data types do not have sizes
- For instance p_sal is NUMBER, not NUMBER(6,2)

ORACLE Academy

Actual Parameters

- Actual parameters can be literal values, variables, or expressions that are sent to the parameter list of a called subprogram
- In the following example, a call is made to raise_sal, where the a_emp_id variable is the actual parameter for the p_id formal parameter, and 100 is the argument (the actual passed value)

```
a_emp_id := 100;
raise_sal(a_emp_id, 2000);
```

ORACLE Academy

Because 2000 is a literal, not the name of a variable, it is both the actual parameter and the argument.

Actual Parameters

- Actual parameters:
  - Are associated with formal parameters during the subprogram call
  - Can also be expressions, as in the following example:

```
raise_sal(a_emp_id,v_raise + 100);
```

ORACLE Academy

Formal and Actual Parameters

- 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

ORACLE Academy

Formal and Actual Parameters

- For instance, you can pass in a salary of '1000.00' in single quotes, so it is coming in as the letter 1 and the letters zero, etc., which get converted into the number one thousand
- This is slower and should be avoided if possible
- You can find out the data types that are expected by using the command DESCRIBE proc_name

ORACLE Academy

Terminology

- Key terms used in this lesson included:
  - Actual parameter
  - Argument
  - Formal parameter
  - Parameters

ORACLE Academy

- Argument – The actual value assigned to a parameter.
- Actual Parameter – Can be literal values, variables, or expressions that are provided in the parameter list of a called subprogram.
- Formal Parameter – A parameter name declared in the procedure heading.
- Parameters – Pass or communicate data between the caller and subprogram and are commonly referred to as arguments.

Summary

- In this lesson, you should have learned how to:
  - Describe how parameters contribute to a procedure
  - Define a parameter
  - Create a procedure using a parameter
  - Invoke a procedure that has parameters
  - Differentiate between formal and actual parameters

ORACLE Academy

ORACLE

Academy

ORACLE

Academy

Database Programming with PL/SQL

8-3

Passing Parameters

ORACLE
Academy

Objectives

- This lesson covers the following objectives:
  - List the types of parameter modes
  - Create a procedure that passes parameters
  - Identify three methods for passing parameters
  - Describe the DEFAULT option for parameters

ORACLE Academy

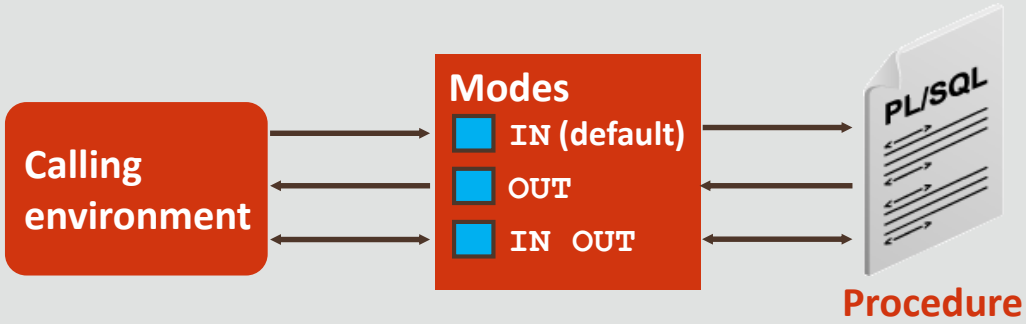
Purpose

- To make procedures more flexible, it is important that varying data is either calculated or passed into a procedure by using input parameters
- Calculated results can be returned to the caller of a procedure by using OUT or IN OUT parameters

Procedural Parameter Modes

- Parameter modes are specified in the formal parameter declaration, after the parameter name and before its data type
- Parameter-passing modes:
  - An IN parameter (the default) provides values for a subprogram to process
  - An OUT parameter returns a value to the caller
  - An IN OUT parameter supplies an input value, which can be returned (output) as a modified value

What are Parameters?



ORACLE Academy

Default Mode: IN

- The IN mode is the default if no mode is specified
- IN parameters can only be read within the procedure
- They cannot be modified

```
CREATE PROCEDURE procedure(param [mode] datatype)
...
```

```
CREATE OR REPLACE PROCEDURE raise_salary
  (p_id      IN my_employees.employee_id%TYPE,
   p_percent IN NUMBER)
IS
BEGIN
  UPDATE my_employees
    SET    salary = salary * (1 + p_percent/100)
    WHERE  employee_id = p_id;
END raise_salary;
```

ORACLE Academy

Using OUT Parameters: 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;
```

```
DECLARE
  a_emp_name employees.last_name%TYPE;
  a_emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(178, a_emp_name, a_emp_sal); ...
END;
```

ORACLE Academy

Using the Previous OUT Example

- Create a procedure with OUT parameters to retrieve information about an employee
- The procedure accepts the value 178 for employee ID and retrieves the name and salary of the employee with ID 178 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 code box at the bottom of the previous slide

Using the Previous OUT Example

- The procedure accepts an employee ID value through the p_id parameter
- The a_emp_name and a_emp_sal variables are populated with the information retrieved from the query into their two corresponding OUT parameters
- Make sure that the data type for the actual parameter variables used to retrieve values from OUT parameters has a size large enough to hold the data values being returned

Viewing OUT Parameters in Application Express

- Use PL/SQL variables that are displayed with calls to the DBMS_OUTPUT.PUT_LINE procedure

```
DECLARE
  a_emp_name employees.last_name%TYPE;
  a_emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(178, a_emp_name, a_emp_sal);
  DBMS_OUTPUT.PUT_LINE('Name: ' || a_emp_name);
  DBMS_OUTPUT.PUT_LINE('Salary: ' || a_emp_sal);
END;
```

```
Name: Grant
Salary: 7700
```

ORACLE Academy

Using IN OUT Parameters: Example

Calling environment

p_phone_no (before the call)

'8006330575'

p_phone_no (after the call)

'(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;
```

ORACLE Academy

Formal parameters must be declared with datatypes, but no explicit sizes. In the slide example, p_phone_no is declared as VARCHAR2, not VARCHAR2(n). The only exception to this rule is when a parameter's datatype is declared implicitly, for example when using %TYPE.

Using the Previous IN OUT Example

- Using an IN OUT parameter, you can pass a value into a procedure that can be updated within the procedure
- The actual parameter value supplied from the calling environment can return as either of the following:
  - The original unchanged value
  - A new value that is set within the procedure

ORACLE Academy

Using the Previous IN OUT Example

- The example in the previous slide creates a procedure with an IN OUT 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’

Calling the Previous IN OUT Example

- The following code creates an anonymous block that declares a_phone_no, assigns the unformatted phone number to it, and passes it as an actual parameter to the FORMAT_PHONE procedure
- The procedure is executed and returns an updated string in the a_phone_no variable, which is then displayed

```
DECLARE
  a_phone_no VARCHAR2(13);
BEGIN
  a_phone_no := '8006330575' ;
  format_phone(a_phone_no);
  DBMS_OUTPUT.PUT_LINE('The formatted number is: ' || a_phone_no);
END;
```

ORACLE Academy

Summary of Parameter Modes

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, constant, expression, 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 Academy

Syntax for Passing Parameters

- There are three ways of passing parameters from the calling environment:
  - Positional: Lists the actual parameters in the same order as the formal parameters (most common method)
  - Named: Lists the actual parameters in arbitrary order and uses the association operator (‘=>' which is an equal and an arrow together) to associate a named formal parameter with its actual parameter
  - Combination: Lists some of the actual parameters as positional (no special operator) and some as named (with the => operator)

Parameter Passing: Examples

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

- Passing by positional notation

```
add_dept ('EDUCATION', 1400);
```

- Passing by named notation

```
add_dept (p_loc=>1400, p_name=>'EDUCATION');
```

ORACLE Academy

Parameter Passing: Examples

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

- Passing by combination notation

```
add_dept ('EDUCATION', p_loc=>1400);
```

- Note : If Combination notation is used, the positional parameters must come first before the named parameters

ORACLE Academy

Parameter Passing

- Will the following call execute successfully?

```
add_dept (p_loc => 1400, 'EDUCATION');
```

- Answer: No, because when using the combination notation, positional notation parameters must be listed before named notation parameters

ORACLE Academy

Parameter Passing

- Will the following call execute successfully?

```
add_dept ('EDUCATION');
```

```
ORA-06550:  line 2, column 1:
PLS-00306:  wrong number or types of arguments in call to
    'ADD_DEPT'
ORA-06550:  line 2, column 1:
PL/SQL:  Statement ignored
1.  begin
2.  add_dept('EDUCATION');
3.  end;
```

- Answer: No
- You must provide a value for each parameter unless the formal parameter is assigned a default value

Parameter Passing Example

- The following procedure with three parameters may be called in the following ways:

```
CREATE OR REPLACE PROCEDURE show_emps (p_emp_id IN NUMBER,
p_department_id IN NUMBER, p_hiredate IN DATE)…
```

- Positional notation :

```
show_emps (101, 10, '01-dec-2006')
```

- Named notation :

```
show_emps(p_department_id => 10,
      p_hiredate => '01-dec-1007', p_emp_id => 101
```

- Combination notation :

```
show_emps(101, p_hiredate => '01-dec-2007',
      p_department_id = 10)
```

Using the DEFAULT Option for IN Parameters

- You can assign a default value for formal IN parameters
- This provides flexibility when passing parameters

```
CREATE OR REPLACE PROCEDURE add_dept(
  p_name my_depts.department_name%TYPE := 'Unknown',
  p_loc  my_depts.location_id%TYPE DEFAULT 1400)
IS
BEGIN
  INSERT INTO my_depts (...)
    VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_dept;
```

- Using the DEFAULT keyword makes it easier to identify that a parameter has a default value

ORACLE Academy

Using the DEFAULT Option for IN Parameters

- The code on the previous slide shows two ways of assigning a default value to an IN parameter
- The two ways shown use:
  - The assignment operator (:=), as shown for the p_name parameter
  - The DEFAULT keyword option, as shown for the p_loc parameter

ORACLE Academy

Using the DEFAULT Option for Parameters

- On the following slide, three ways of invoking the add_dept procedure are displayed:
  - The first example uses 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 and the supplied value for the p_loc parameter

ORACLE Academy

Note: In Application Express, DESCRIBE procedure-name does not show which parameters have default values.

Using the DEFAULT Option for Parameters

- Referring to the code on Slide #21, we know the add_dept procedure has two IN parameters and both parameters have default values

```
add_dept;
add_dept ('ADVERTISING', p_loc => 1400);
add_dept (p_loc => 1400);
```

ORACLE Academy

The named notation for passing parameters is especially useful when the called procedure contains many IN parameters, many of which have default values. Imagine a procedure which declares 10 İN parameters, all of which have default values. We want to call the procedure, overriding the default value only for the last parameter. We can simply code:

…. procedure_name(last_parameter_name => value);

Using the positional notation, all ten values must be passed, and the default values could not be used.

Guidelines for Using the DEFAULT Option for Parameters

- You cannot assign default values to OUT and IN OUT parameters in the header, but you can in the body of the procedure
- 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
- A parameter inheriting a DEFAULT value is different from NULL

ORACLE Academy

Working with Parameter Errors During Runtime

- 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:

```
BEGIN
  add_dept(name =>'new dept', 'new location');
END;
```

- The following error message is generated:

```
ORA-06550:  line 2, column 3:
PLS-00306:  a positional parameter association may not follow a named association
ORA-06550:  line 2, column 3:
PL/SQL:  Statement ignored
1.   BEGIN
2.         add_dept(name=>'new dept', 'new location');
3.   END;
```

Terminology

- Key terms used in this lesson included:
  - Combination Notation
  - IN parameter
  - IN OUT parameter
  - Named Notation
  - OUT parameter
  - Positional Notation

ORACLE Academy

PLSQL 8-3
Passing Parameters

- Combination Notation - Lists some of the actual parameters as positional (no special operator) and some as named (with the => operator).
- IN Parameter – Provides values for a subprogram to process.
- IN OUT Parameter – Supplies an input value, which may be returned as a modified value.
- Named Notation - Lists the actual parameters in arbitrary order and uses the association operator (‘=>' which is an equal and an arrow together) to associate a named formal parameter with its actual parameter.
- OUT Parameter – Returns a value to the caller.
- Positional Notation - Lists the actual parameters in the same order as the formal parameters.

Summary

- In this lesson, you should have learned how to:
  - List the types of parameter modes
  - Create a procedure that passes parameters
  - Identify three methods for passing parameters
  - Describe the DEFAULT option for parameters

ORACLE Academy

ORACLE

Academy