Creating Stored Procedures

Objectives

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

- Describe and create a procedure
- Create procedures with parameters
- Differentiate between formal and actual parameters
- Use different parameter-passing modes
- Invoke a procedure
- Handle exceptions in procedures
- Remove a procedure

What are PL/SQL Subprograms?

- Is a named PL/SQL block that can be called with a set of parameters
- A subprogram consists of a specification and a body
- Can be a stored procedure or a function
- You use a procedure to perform an action and a function to compute and return a value

Benefits of PL/SQL Subprograms

- Easy maintenancy: located in one place
- Improved data security: only users with the right privileges can execute them
- Data integrity: related actions are performed together or not at all
- Improved performance: less network traffic
- Improved code clarity: can be attained by using appropriate names and conventions to describe the action of the routines

Differences between anonymous blocks and subprograms

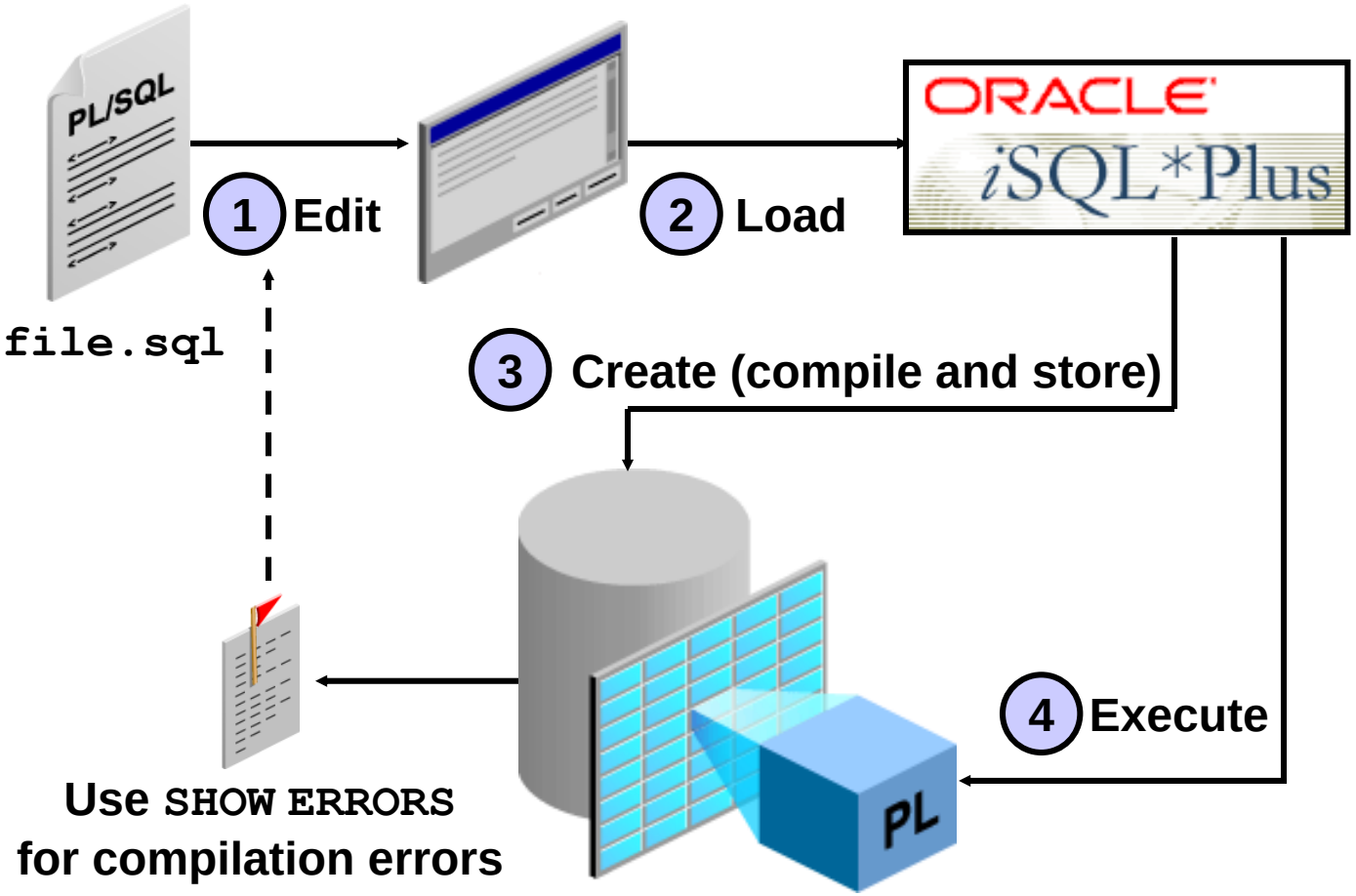
Anonymous Blocks	Subprograms
Unnamed PL/SQL blocks	Named PL/SQL blocks
Compiled every time	Compiled only once
Not stored in the database	Stored in the database
Cannot be invoked by other applications	Named and, therefore, can be invoked by other applications
Do not return values	Subprograms called functions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

What Is a Procedure?

A procedure:

- **Is a type of subprogram that performs an action**
- **Can be stored in the database as a schema object**
- **Promotes reusability and maintainability**

Developing Procedures



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

Example procedure without parameters
(from PL/SQL fundamentals lesson 9)

```
...
CREATE TABLE dept AS SELECT * FROM departments;
CREATE 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;
```

/ → creation of the Procedure

The source code is saved in de data dictionary

When the code is executed successfully: a compiled version is stored in the database

When the code contains errors:

- **Message: ‘created with compilation errors’.**
- **List the errors by typing: show errors**

Invoking the Procedure

```
BEGIN
  add_dept;
END;
```

OR: execute add_dept

exec add_dept

Klassikaal voorbeeld

What Are Parameters?

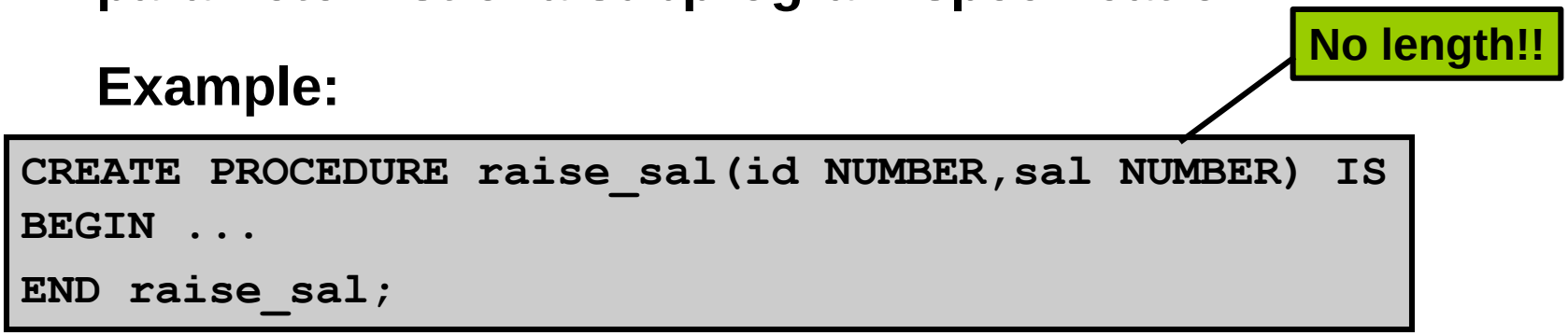
Parameters:

- **Are declared after the subprogram name in the PL/SQL header**
- **Pass or communicate data between the caller and the subprogram**
- **Are used like local variables but are dependent on their parameter-passing mode:**
  - **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 may be returned (output) as a modified value.**

Formal and Actual Parameters

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

  Example:



- **Actual parameters: Literal values, variables, or expressions used in the parameter list of the called subprogram**

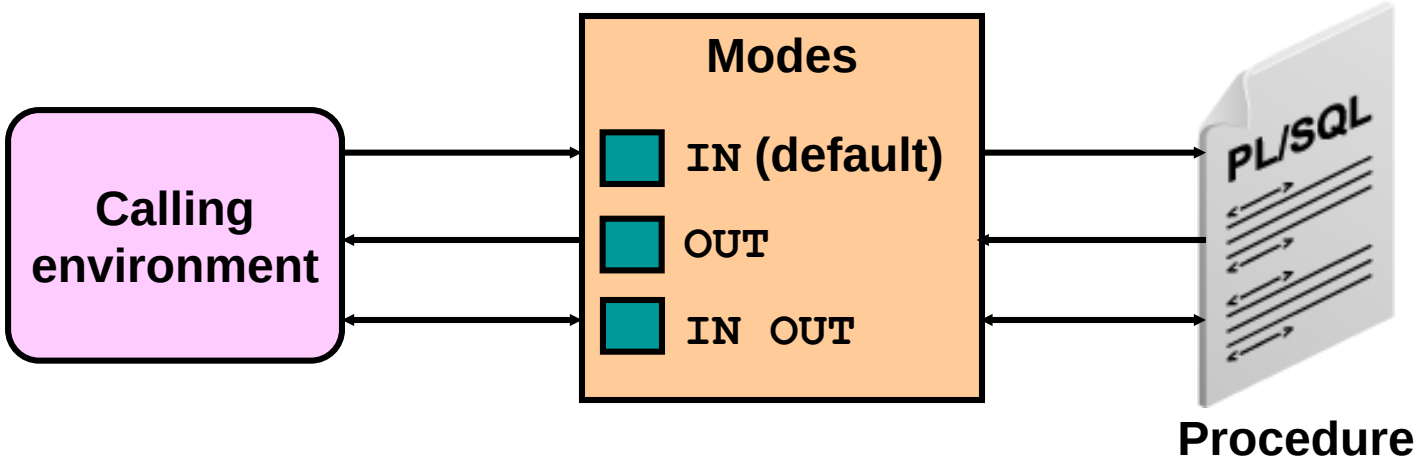
  Example:

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

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.

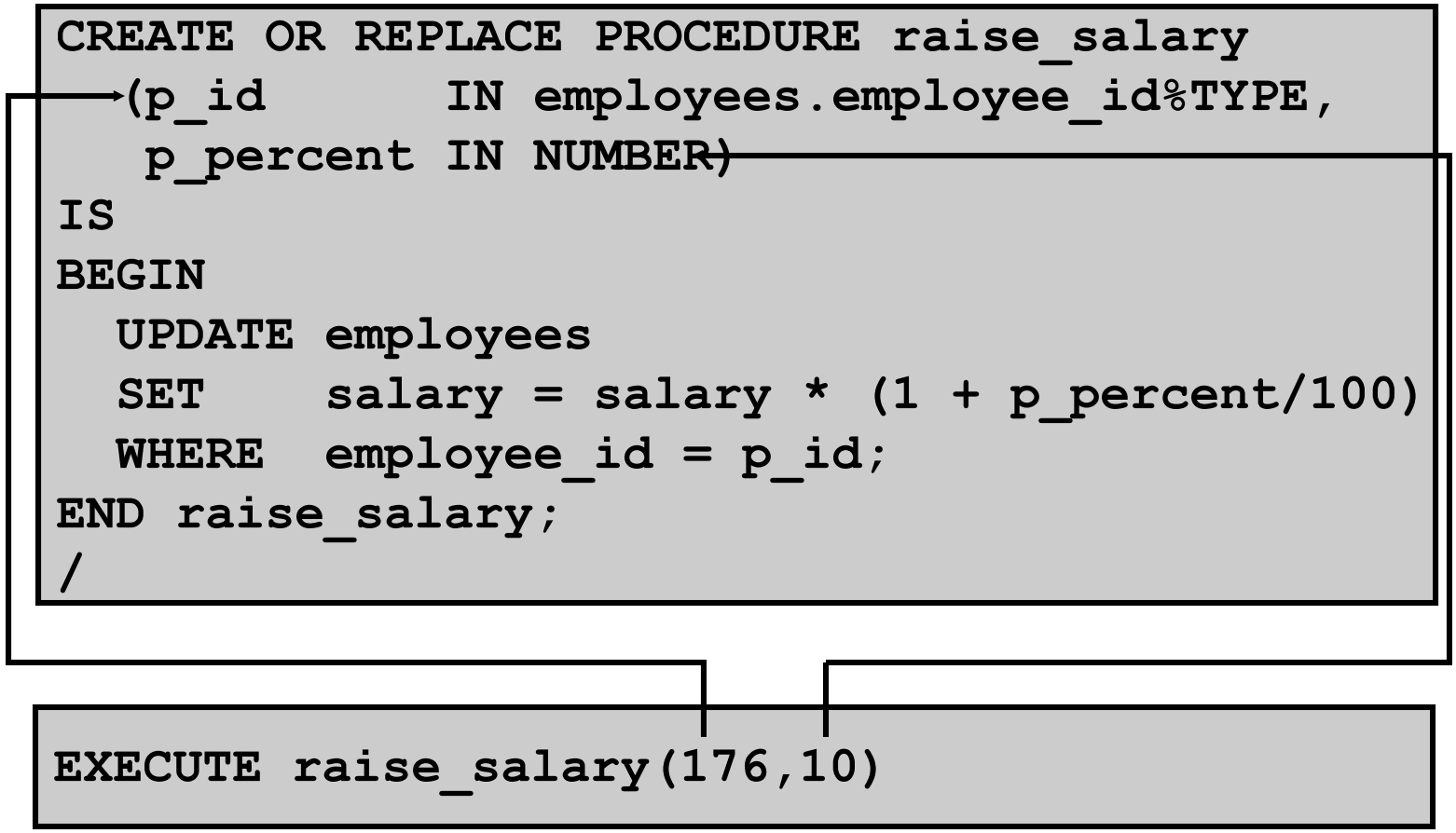
```
CREATE PROCEDURE procedure(param [mode] datatype)
...
```



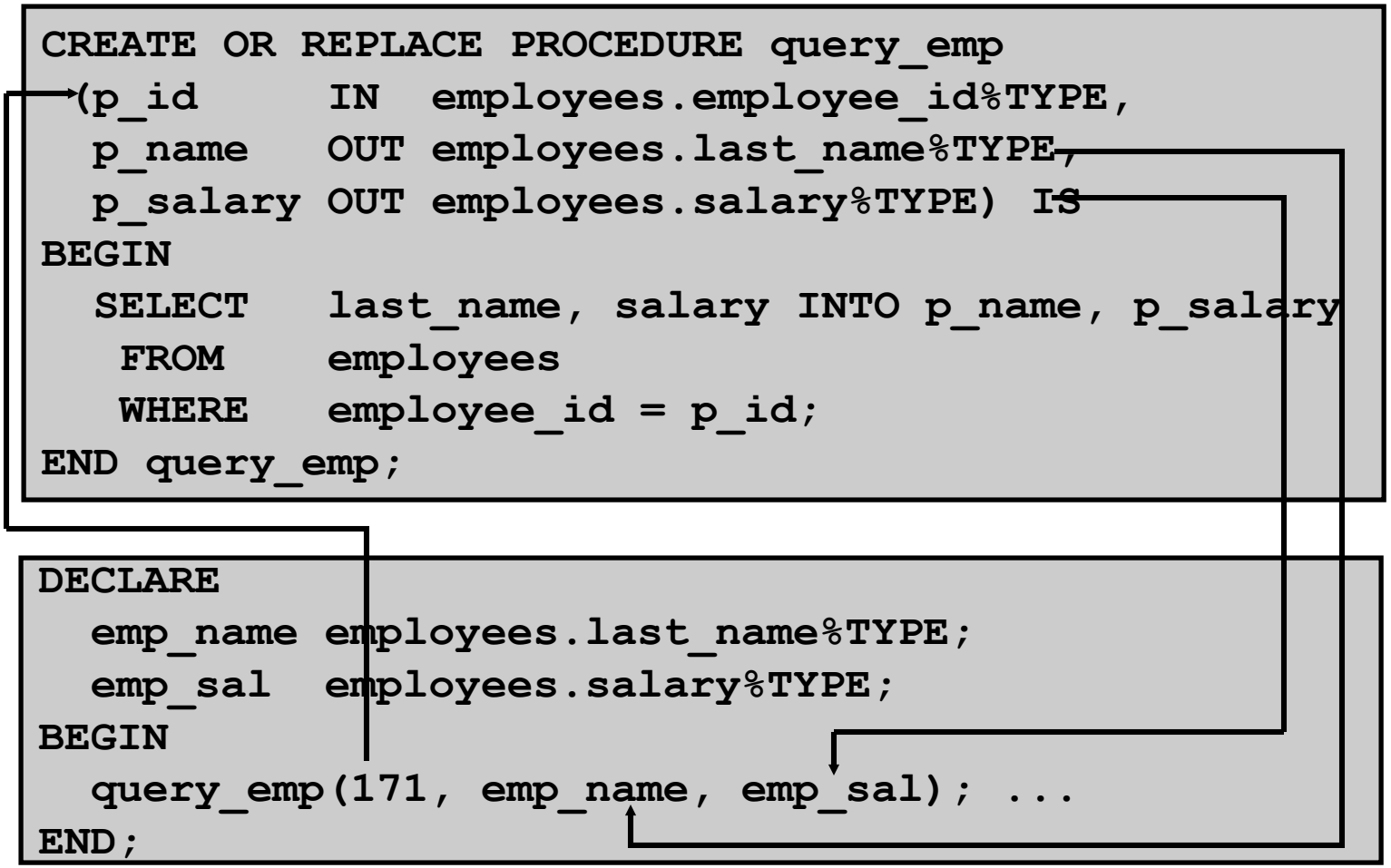
Summary of Parameter Modes

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, 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

Using IN Parameters: Example



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
  emp_name employees.last_name%TYPE;
  emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(171, emp_name, emp_sal); ...
END;
```

Viewing OUT Parameters with *i*SQL*Plus

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

```
SET SERVEROUTPUT ON
DECLARE
  emp_name employees.last_name%TYPE;
  emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(171, emp_name, emp_sal);
  DBMS_OUTPUT.PUT_LINE('Name: ' || emp_name);
  DBMS_OUTPUT.PUT_LINE('Salary: ' || emp_sal);
END;
```

- Use *i*SQL*Plus host variables, execute QUERY_EMP using host variables, and print the host variables.

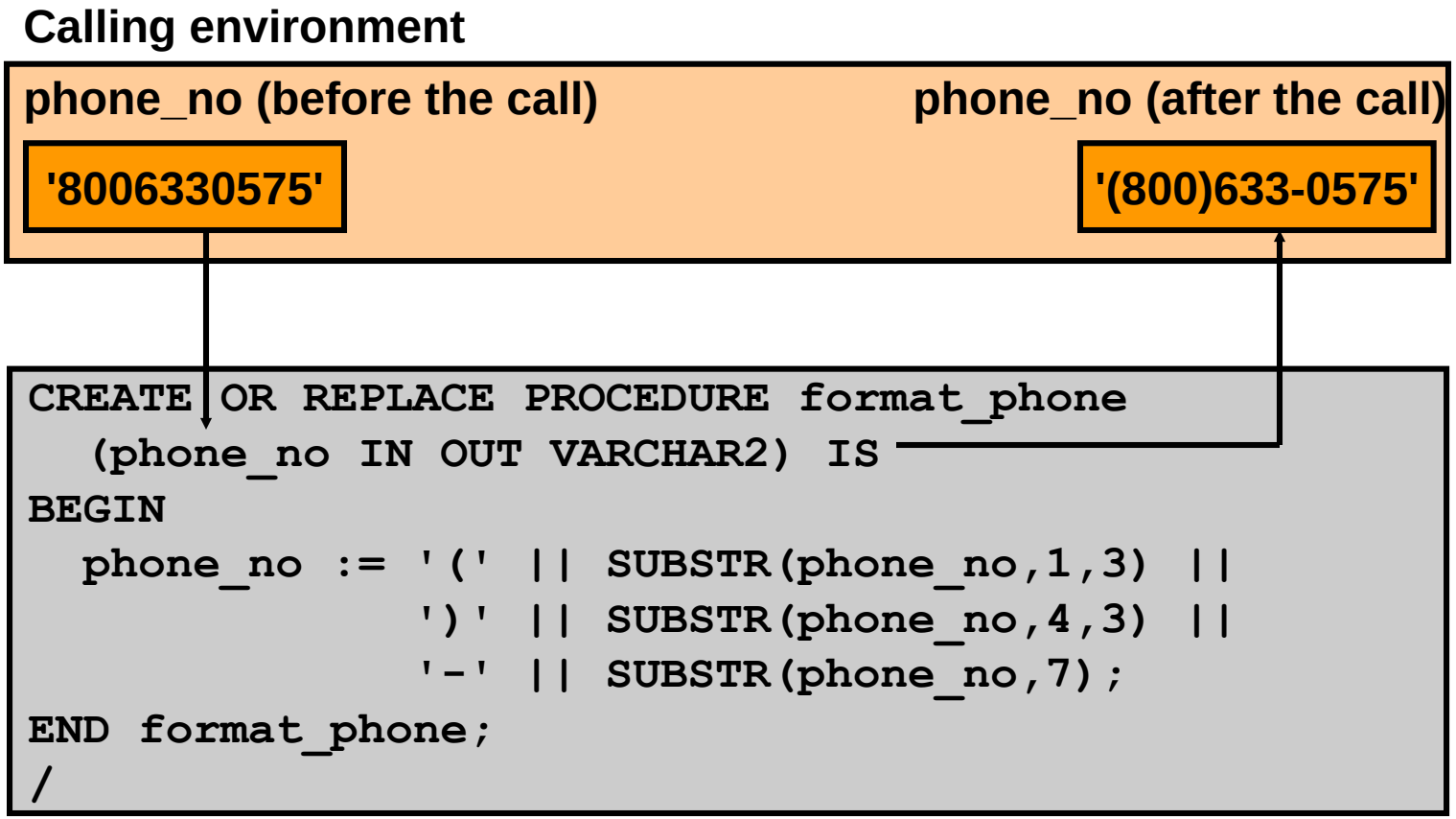
```
VARIABLE name VARCHAR2(25)
VARIABLE sal  NUMBER
EXECUTE query_emp(171, :name, :sal)
PRINT name sal
```

Calling PL/SQL Using Host Variables

A host variable (also known as a bind or a global variable):

- Is declared and exists externally to the PL/SQL subprogram. A host variable can be created in:
  - *i*SQL*Plus by using the `VARIABLE` command
  - Oracle Forms internal and UI variables
  - Java variables
- Is preceded by a colon (:) when referenced in PL/SQL code
- Can be referenced in an anonymous block but not in a stored subprogram
- Provides a value to a PL/SQL block and receives a value from a PL/SQL block

Using IN OUT Parameters: Example



Klassikaal voorbeeld

Syntax for Passing Parameters

- **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**
- **Combination:**
  - **Lists some of the actual parameters as positional and some as named**

Parameter Passing: Examples

```
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;
/
```

- Passing by positional notation

```
EXECUTE add_dept ('TRAINING', 2500)
```

- Passing by named notation

```
EXECUTE add_dept (p_loc=>2400, p_name=>'EDUCATION')
```

Hoe moet in opgave2 de procedure opgeroepen worden door gebruik te maken van de named notation?

Using the DEFAULT Option for Parameters

- Defines default values for parameters:

```
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 (...)
  VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_dept;
```

- Provides flexibility by combining the positional and named parameter-passing syntax:

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

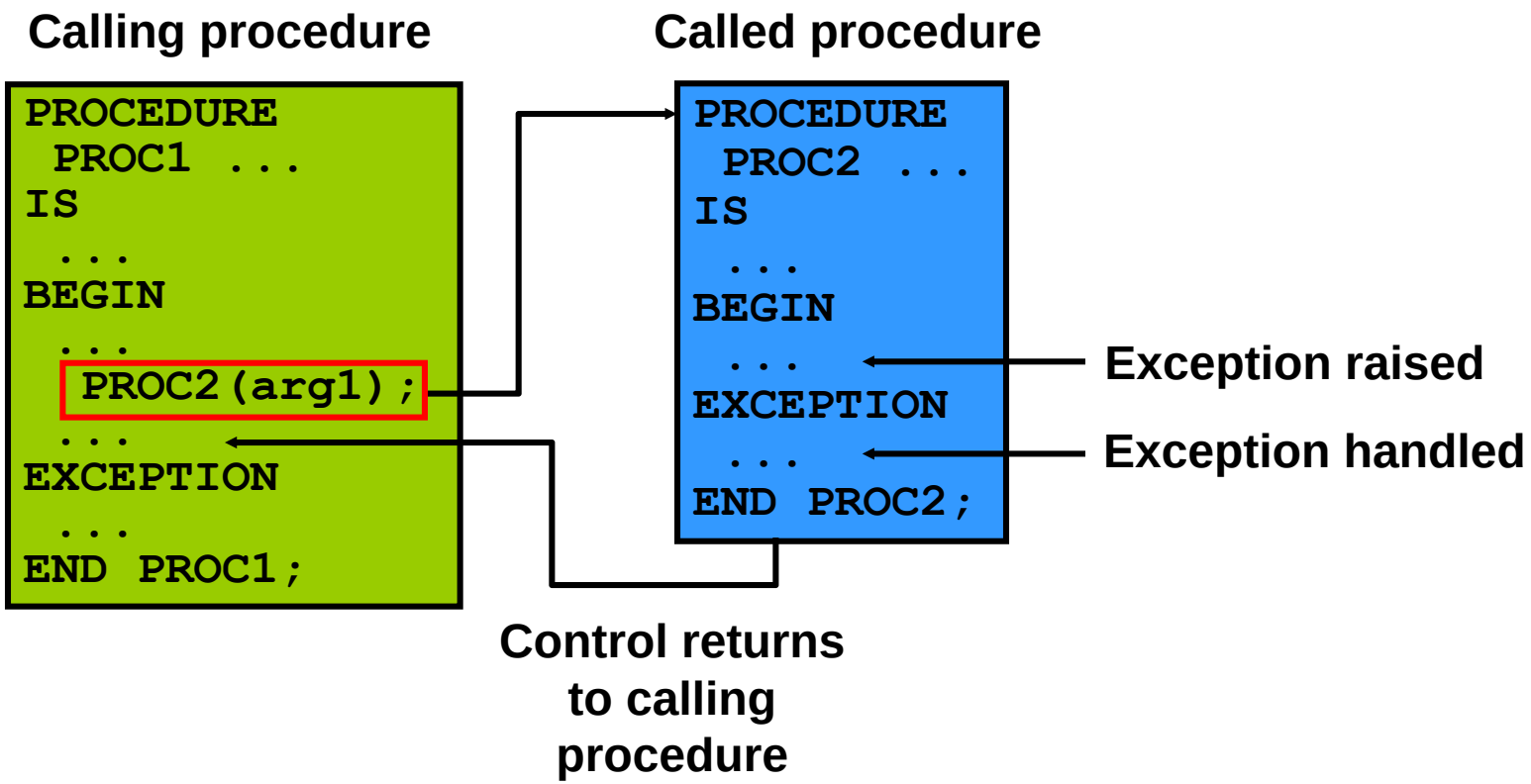
Invoking Procedures

You can invoke parameters by:

- Using anonymous blocks
- Using another procedure, as in the following:

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

Handled Exceptions

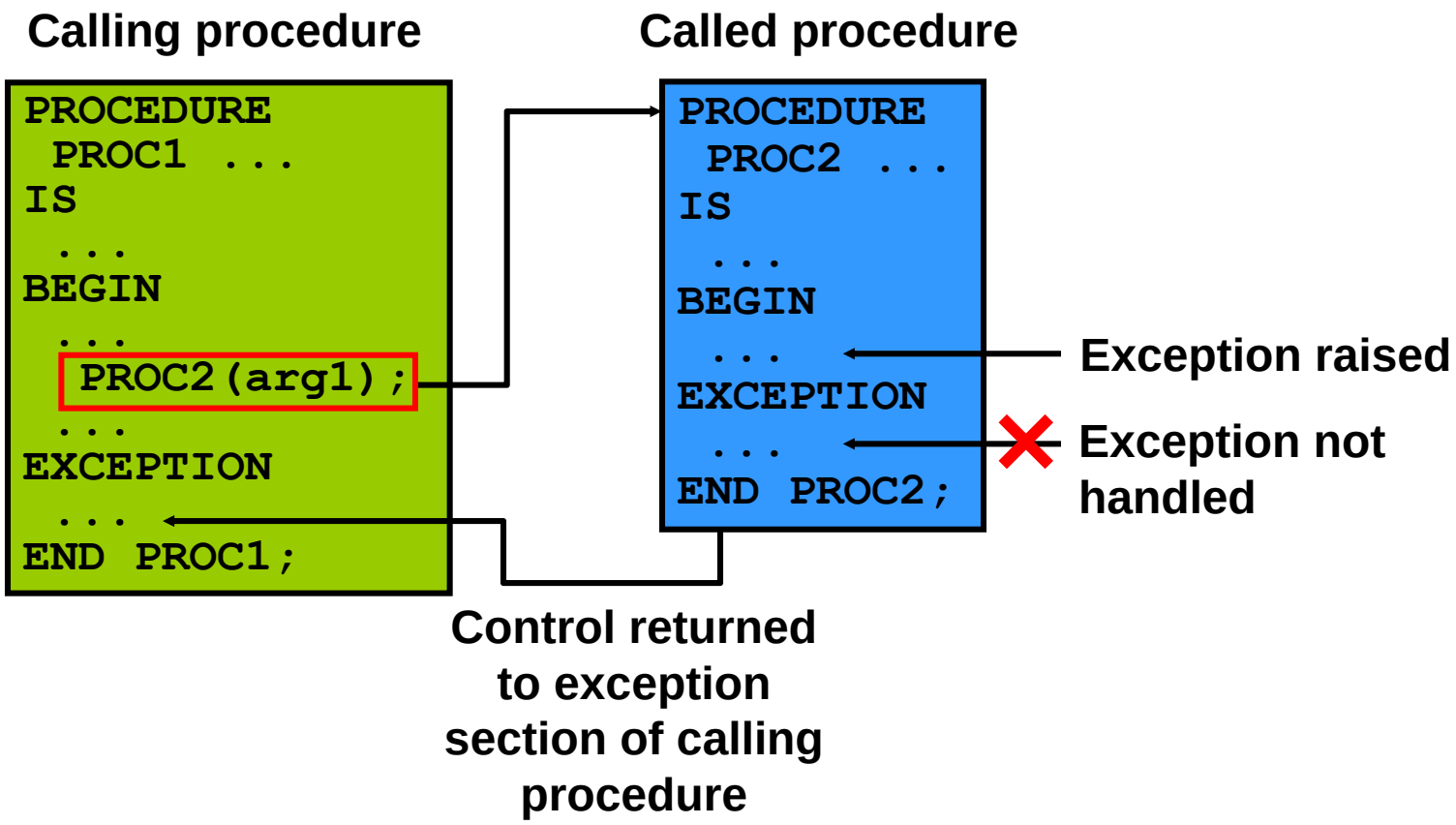


Handled Exceptions: Example

```
CREATE PROCEDURE add_department(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
BEGIN
  INSERT INTO DEPARTMENTS (department_id,
    department_name, manager_id, location_id)
  VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
  DBMS_OUTPUT.PUT_LINE('Added Dept: '||name);
EXCEPTION
 WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Err: adding dept: '||p_name);
END;
```

```
CREATE PROCEDURE create_departments IS
BEGIN
  add_department('Media', 100, 1800);
  add_department('Editing', 99, 1800);
  add_department('Advertising', 101, 1800);
END;
```

Exceptions Not Handled



Exceptions Not Handled: Example

```
CREATE PROCEDURE add_department_noex(
    p_name VARCHAR2, p_mgr NUMBER, p_loc NUMBER) IS
BEGIN
  INSERT INTO DEPARTMENTS (department_id,
    department_name, manager_id, location_id)
  VALUES (DEPARTMENTS_SEQ.NEXTVAL, p_name, p_mgr, p_loc);
  DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_name);
END;
```

```
CREATE PROCEDURE create_departments_noex IS
BEGIN
  add_department_noex('Media', 100, 1800);
  add_department_noex('Editing', 99, 1800);
  add_department_noex('Advertising', 101, 1800);
END;
```

Removing Procedures

You can remove a procedure that is stored in the database.

- **Syntax:**

```
DROP PROCEDURE procedure_name
```

- **Example:**

```
DROP PROCEDURE raise_salary;
```

Viewing Procedures in the Data Dictionary

Information for PL/SQL procedures is saved in the following data dictionary views:

- View source code in the `USER_SOURCE` table to view the subprograms that you own, or the `ALL_SOURCE` table for procedures that are owned by others who have granted you the `EXECUTE` privilege.

```
SELECT text
FROM   user_source
WHERE  name='ADD_DEPARTMENT' and type='PROCEDURE'
ORDER BY line;
```

- View the names of procedures in `USER_OBJECTS`.

```
SELECT object_name
FROM   user_objects
WHERE  object_type = 'PROCEDURE';
```

Benefits of Subprograms

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

Summary

In this lesson, you should have learned how to:

- Write a procedure to perform a task or an action
- Create, compile, and save procedures in the database by using the `CREATE PROCEDURE` SQL command
- Use parameters to pass data from the calling environment to the procedure using three different parameter modes: `IN` (the default), `OUT`, and `IN OUT`
- Recognize the effect of handling and not handling exceptions on transactions and calling procedures

Summary

- Remove procedures from the database by using the `DROP PROCEDURE` SQL command
- Modularize your application code by using procedures as building blocks

Practice 1: Overview

This practice covers the following topics:

- Creating stored procedures to:
  - Insert new rows into a table using the supplied parameter values
  - Update data in a table for rows that match the supplied parameter values
  - Delete rows from a table that match the supplied parameter values
  - Query a table and retrieve data based on supplied parameter values
- Handling exceptions in procedures
- Compiling and invoking procedures