

# 9

## Creating Procedures

# 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**

# 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**

# PL/SQL Program Constructs

Tools Constructs
Anonymous blocks
Application procedures or functions
Application packages
Application triggers
Object types

```
<header> IS | AS  
or DECLARE  
□ □ □  
BEGIN  
□ □ □  
EXCEPTION  
□ □ □  
END;
```

Database Server Constructs
Anonymous blocks
Stored procedures or functions
Stored packages
Database triggers
Object types

# 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**

# 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)

```

CREATE TABLE T1_kiemtra (thongbao varchar2(200));
DECLARE
    v_ename employees.last_name%TYPE;
    v_sal employees.salary%TYPE := &p_sal;
BEGIN
    SELECT last_name INTO v_ename FROM employees WHERE salary = v_sal;
    INSERT INTO T1_kiemtra (thongbao)    VALUES (v_ename || ' - ' || v_sal);
EXCEPTION
    WHEN no_data_found THEN
        INSERT INTO T1_kiemtra (thongbao)
        VALUES ('No employee with a salary of '|| TO_CHAR(v_sal));
    WHEN too_many_rows THEN
        INSERT INTO T1_kiemtra (thongbao)
        VALUES ('More than one employee with a salary of '|| TO_CHAR(v_sal));
    WHEN others THEN
        INSERT INTO T1_kiemtra (thongbao)
        VALUES ('Some other error occurred.');
```

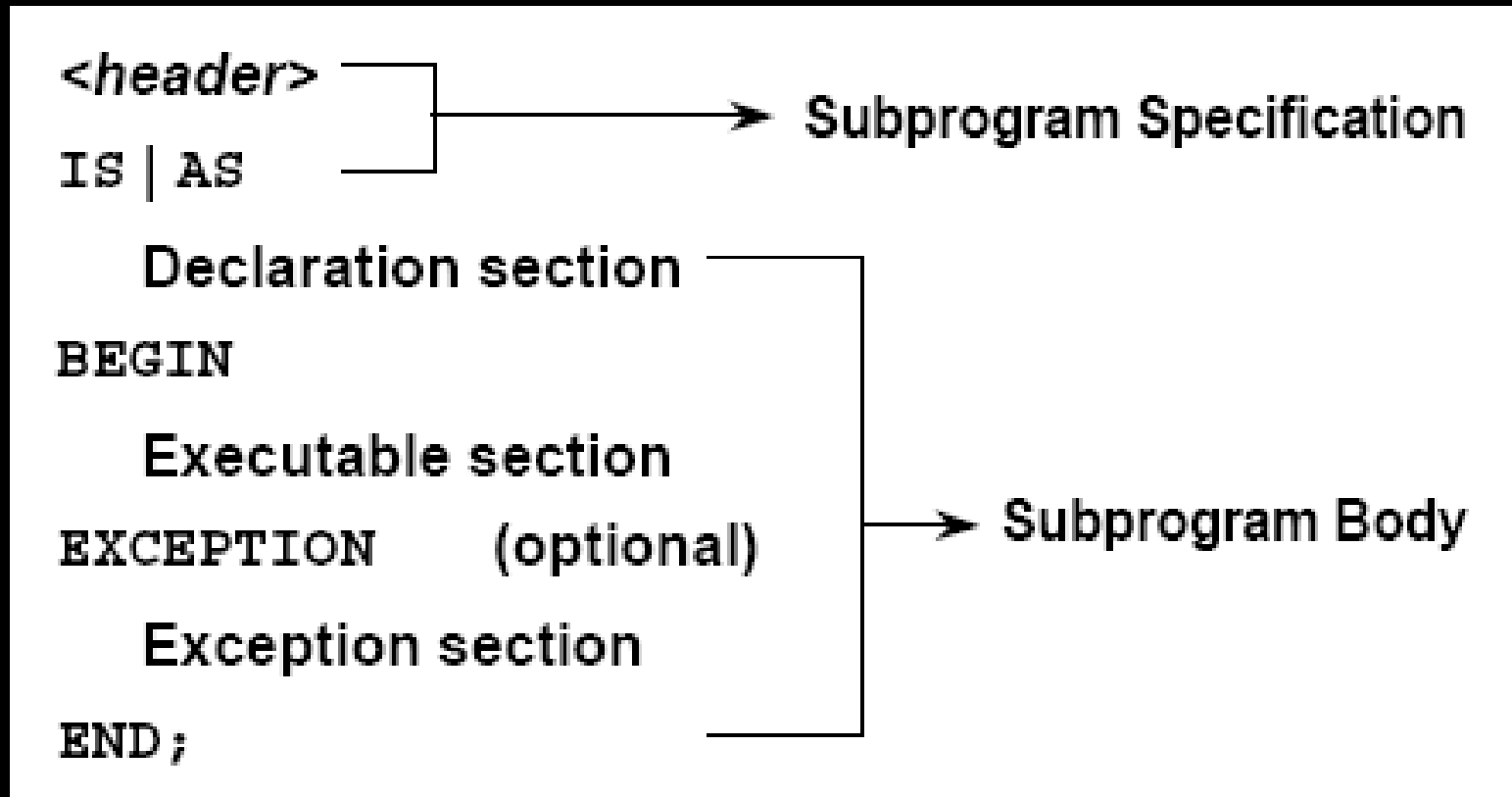
END;

/

```

SELECT * FROM T1_kiemtra;
```

# Block Structure for PL/SQL Subprograms



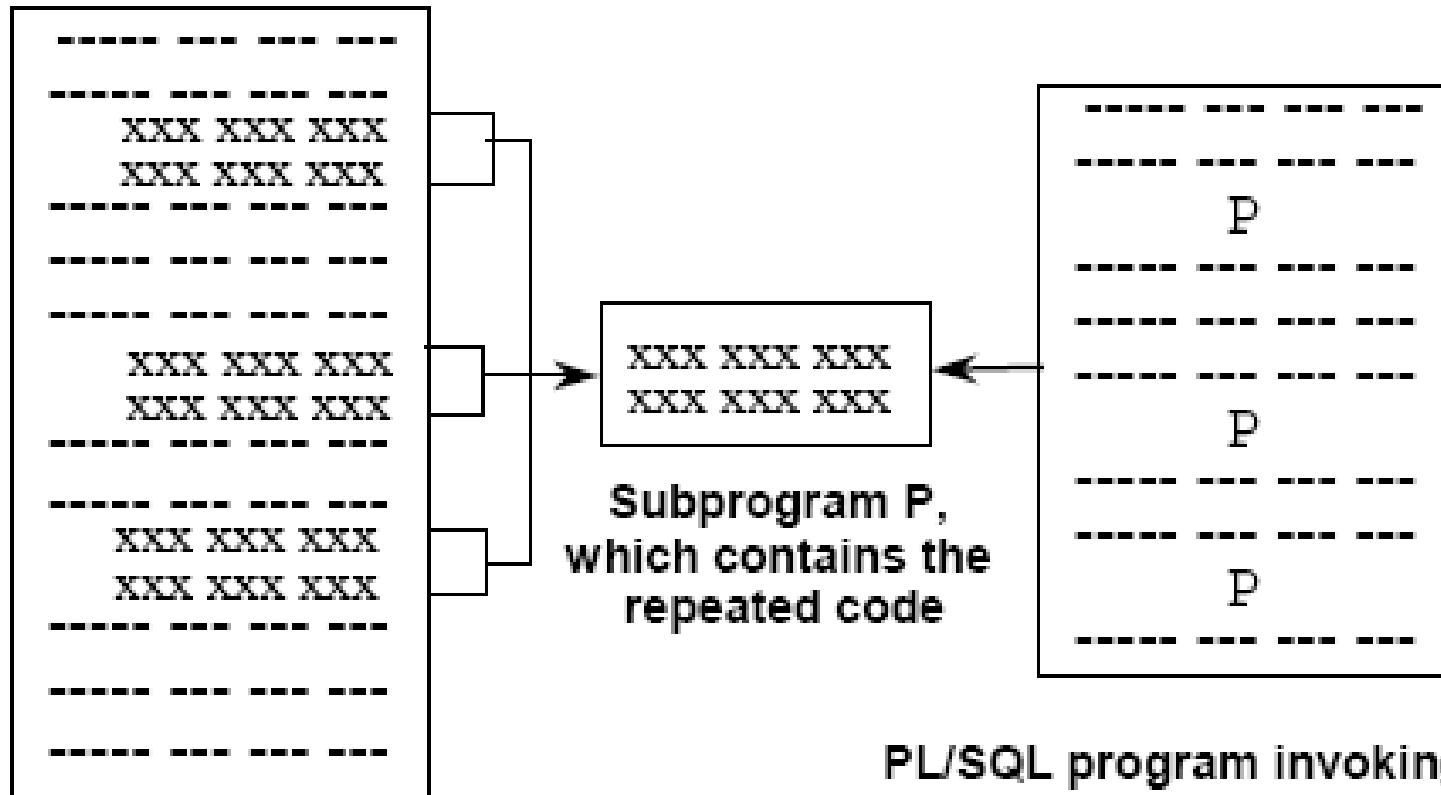


```
CREATE TABLE TEST (  
  MA      NUMBER(4) PRIMARY KEY,  
  THONGBAO  VARCHAR2(100));
```

```
CREATE OR REPLACE PROCEDURE INS_TEST  
(p_ma IN TEST.ma%TYPE, p_thongbao IN TEST.thongbao%TYPE)  
IS  
BEGIN  
  INSERT INTO TEST (ma, thongbao)  
  VALUES (p_ma, p_thongbao);  
  COMMIT;  
END INS_TEST;  
/
```

```
EXECUTE ins_test (1111, 'Database Administrator');  
SELECT * FROM TEST;
```

# PL/SQL Subprograms



Code repeated more than once in a PL/SQL program

PL/SQL program invoking the subprogram at multiple locations

# Benefits of Subprograms

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

# Developing Subprograms by Using *iSQL\*Plus*

The screenshot displays the iSQL\*Plus web interface. At the top, a Notepad window titled 'logexec.sql - Notepad' shows the following SQL code:

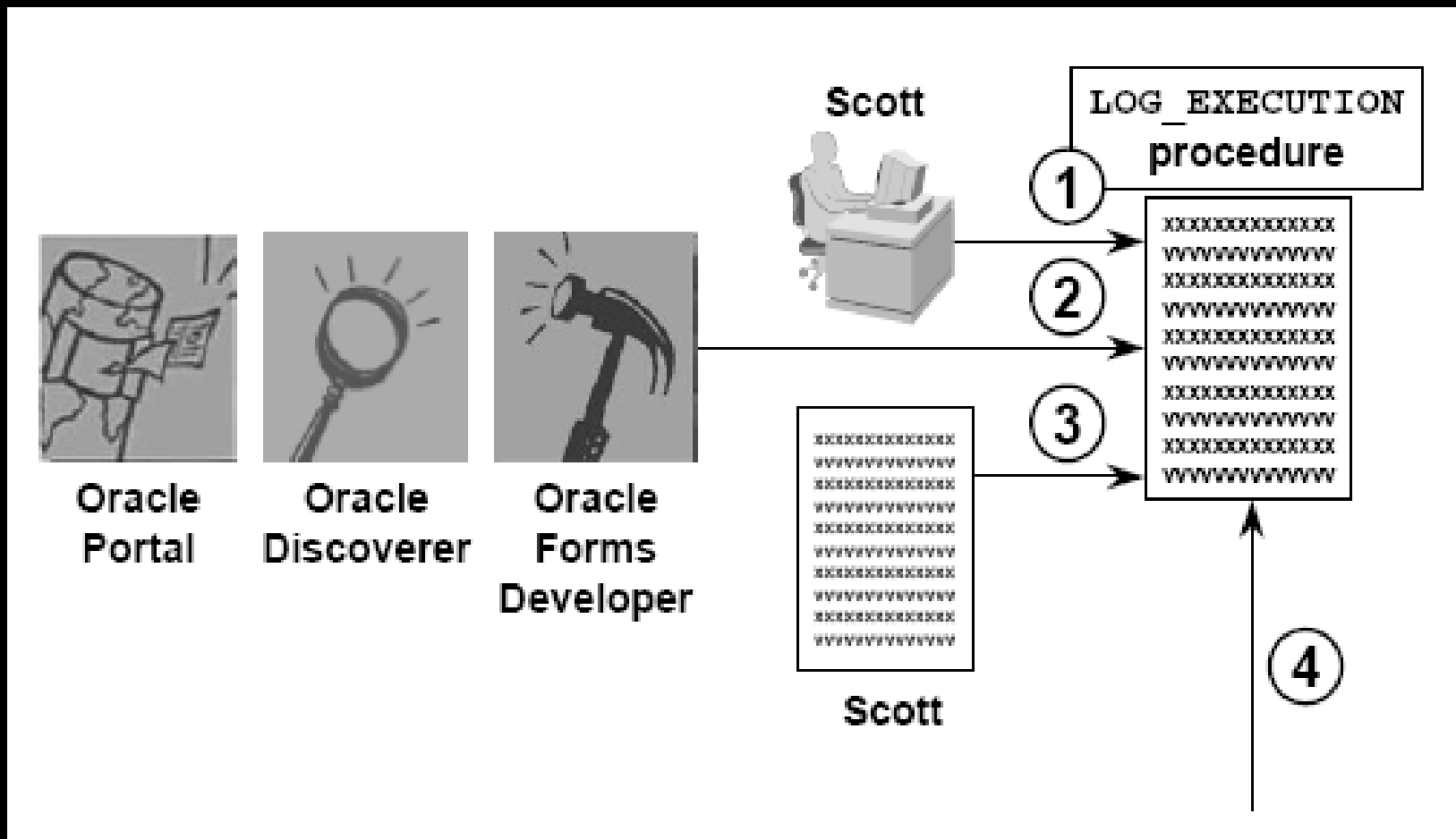
```
CREATE OR REPLACE PROCEDURE log_execution
IS
BEGIN
  INSERT INTO log_table (user_id, log_date)
  VALUES (user, sysdate);
END log_execution;
```

Below the Notepad window, the 'Script Location' field contains 'D:\demo\01\_logexec.sql'. To the right of this field are 'Browse...' and 'Load Script' buttons. Below the 'Script Location' field is the 'Enter statements:' area, which contains the same SQL code as the Notepad window. To the right of the 'Enter statements:' area are 'Execute', 'Clear Screen', and 'Save Script' buttons. The 'Output' dropdown menu is set to 'Work Screen'.

Numbered callouts indicate the following steps:

1. Editing the SQL script in the Notepad window.
2. The 'Script Location' field.
3. The 'Enter statements:' area.
4. The 'Execute' button.

# Invoking Stored Procedures and Functions



# Invoking Stored Procedures and Functions

iSQL*Plus	EXECUTE log_execution
Oracle development tools such as Oracle Forms Developer	log_execution;
Another procedure	<pre>CREATE OR REPLACE PROCEDURE leave_emp (p_id IN employees.employee_id%TYPE) IS BEGIN   DELETE FROM employees     WHERE employee_id = p_id;   log_execution; END leave_emp;</pre>

# 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.**

# 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*.

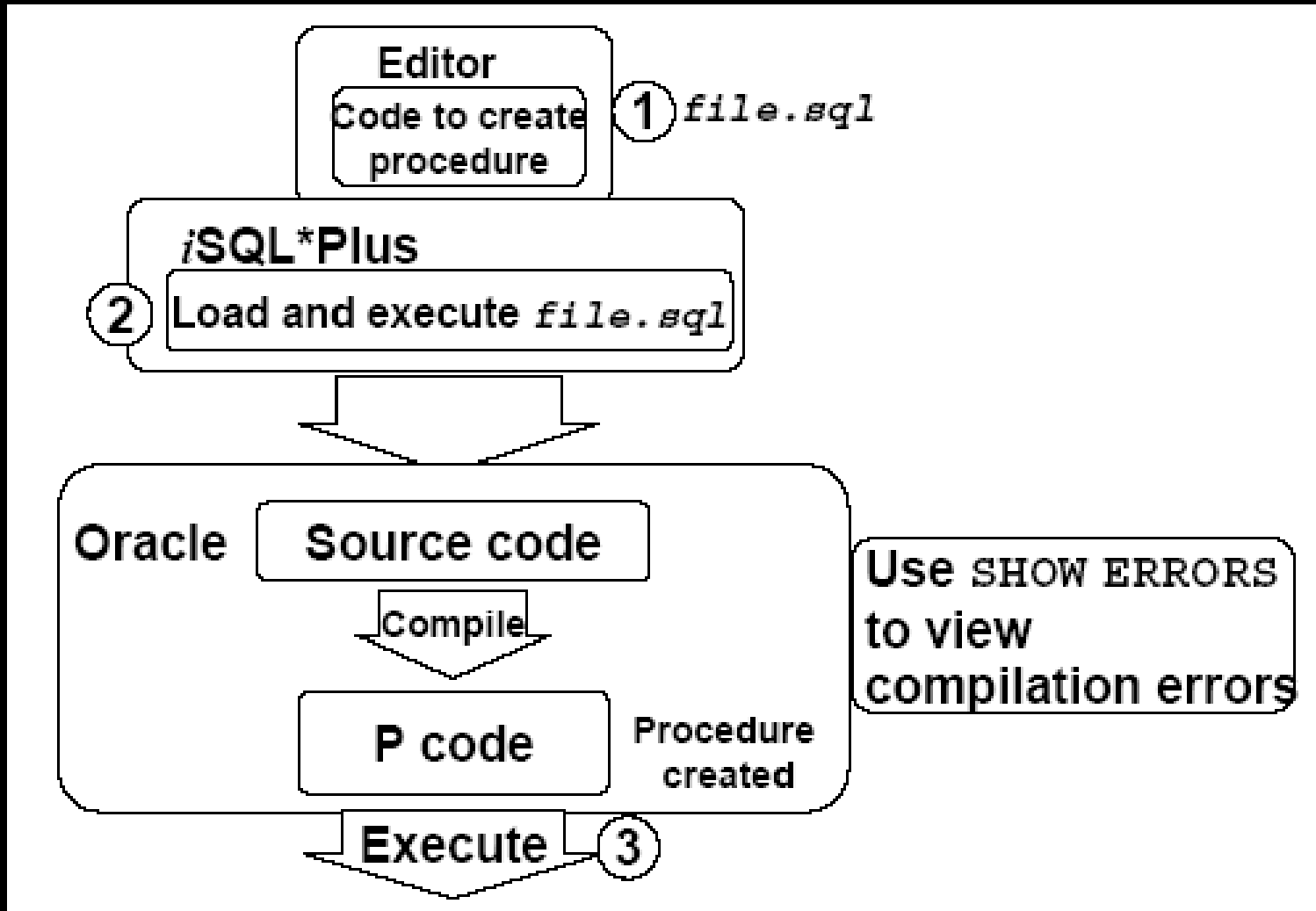


## Syntax for Creating Procedures

### Syntax Definitions

Parameter	Description
<i>procedure_name</i>	Name of the procedure
<i>parameter</i>	Name of a PL/SQL variable whose value is passed to or populated by the calling environment, or both, depending on the <i>mode</i> being used
<i>mode</i>	Type of argument: IN (default) OUT IN OUT
<i>Data type</i>	Data type of the argument—can be any SQL / PLSQL data type. Can be of %TYPE, %ROWTYPE, or any scalar or composite data type.
<i>PL/SQL block</i>	Procedural body that defines the action performed by the procedure

# Developing Procedures



# 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

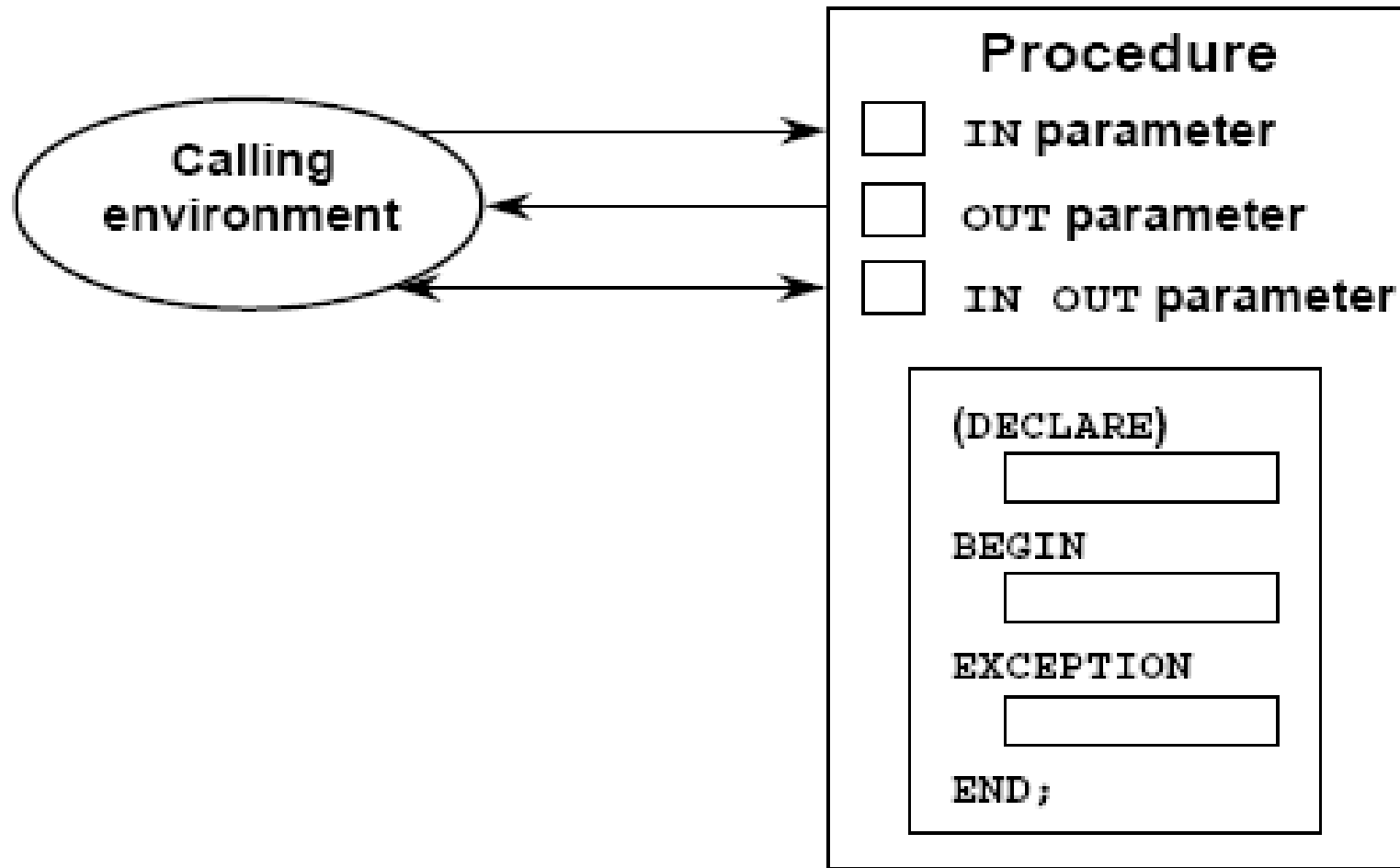
**Example:**

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

```
CREATE OR REPLACE PROCEDURE add_job
(p_jobid IN jobs.job_id%TYPE, p_jobtitle IN jobs.job_title%TYPE)
IS
BEGIN
    INSERT INTO jobs (job_id, job_title)
    VALUES (p_jobid, p_jobtitle);
    COMMIT;
END add_job;
/
EXECUTE add_job ('IT_DBA', 'Database Administrator');

SELECT * FROM jobs WHERE job_id = 'IT_DBA';
```

# Procedural Parameter Modes



# 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

# 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;
/
```

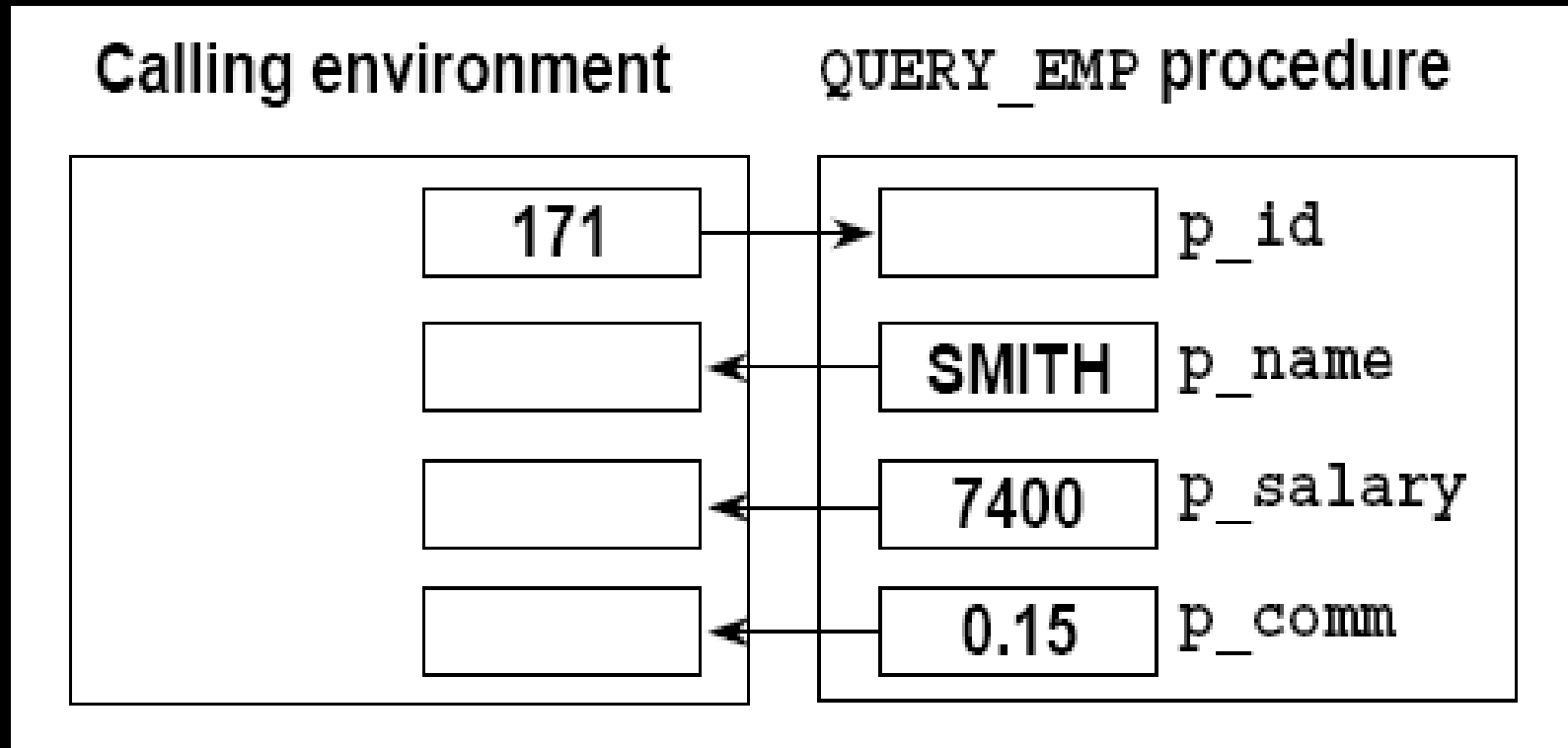
To invoke a procedure in *iSQL\*Plus*, use the EXECUTE command.

```
EXECUTE raise_salary (176)
```

To invoke a procedure from another procedure, use a direct call. At the location of calling the new procedure, enter the procedure name and actual parameters.

```
raise_salary (176);
```

# OUT Parameters: Example





# 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

# 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
```

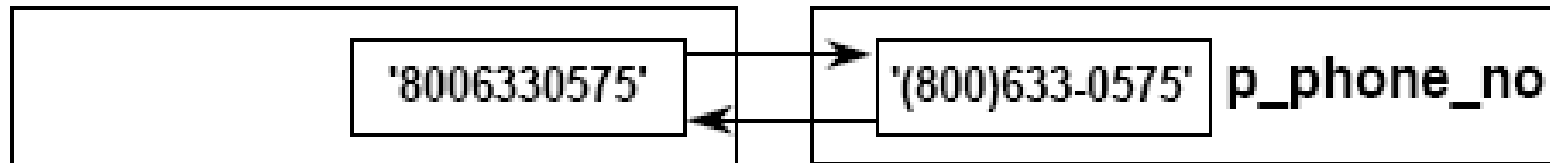
PL/SQL procedure successfully completed

G_NAME
Smith

# IN OUT Parameters

Calling environment

FORMAT\_PHONE procedure



```
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.

# 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

# 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.

# 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**

# 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
...	Human Resources	2400
290	TRAINING	2500
300	EDUCATION	2400
310	unknown	1200

31 rows selected.

# Examples of Passing Parameters

The anonymous block above shows the different ways the ADD\_DEPT procedure can be invoked, and the output of each way the procedure is invoked. Usually, you can use positional notation to override the default values of formal parameters.

However, you cannot skip a formal parameter by leaving out its actual parameter. Note: All the positional parameters should precede the named parameters in a subprogram call.

Otherwise, you will receive an error message, as shown in the following example:

```
EXECUTE add_dept(p_name=>'new dept', 'new location')
```

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

```
*
```

```
ERROR at line 1:
```

```
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
```



# Declaring Subprograms

## leave\_emp2.sql

```
create table log_table ( user_id varchar2(20),
                        log_date date);
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;
/
```

# Invoking a Procedure from an Anonymous PL/SQL Block

```
DECLARE
    v_id NUMBER := 163;
BEGIN
    leave_emp2 (v_id);      --invoke procedure
    COMMIT;

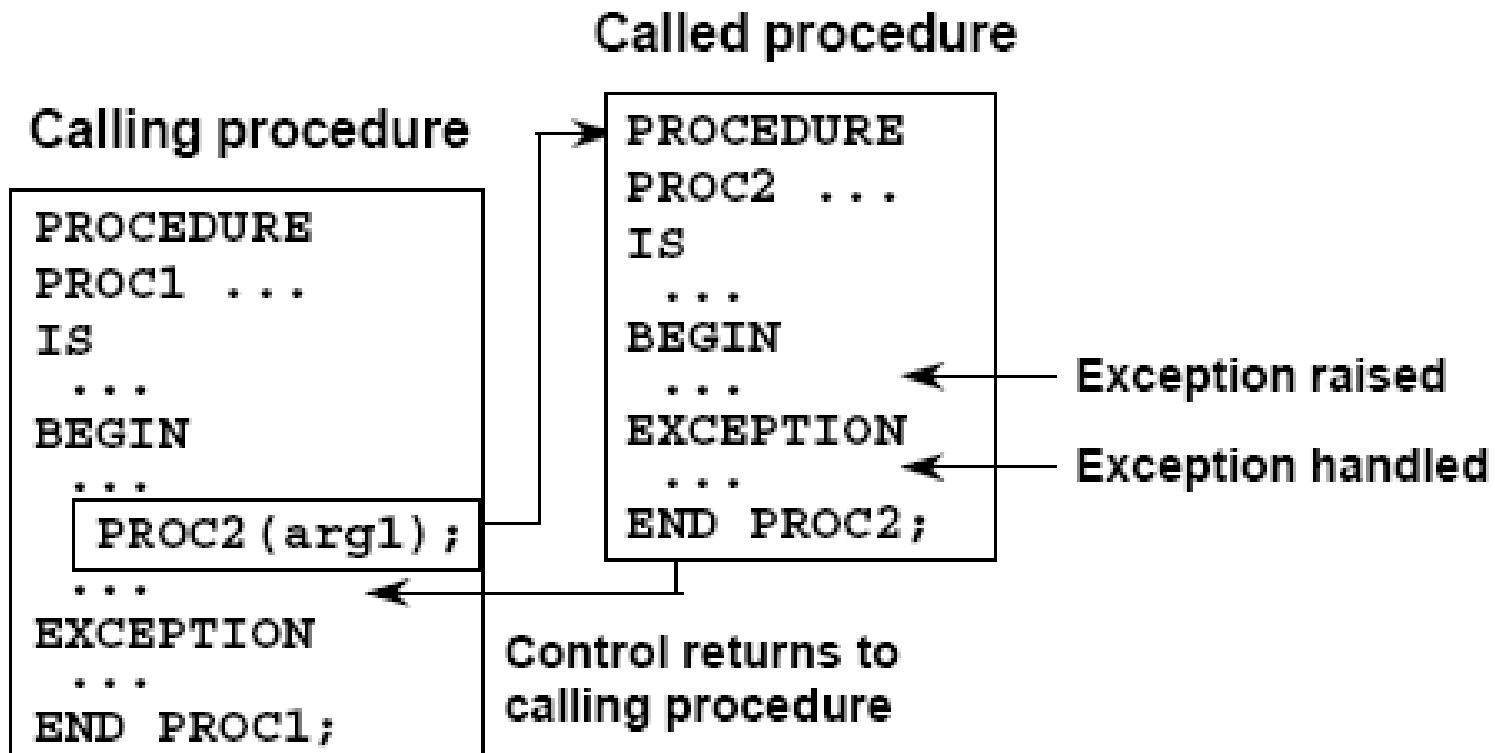
...
END;
```

# 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
        WHERE department_id = 20;
BEGIN
    FOR emp_rec IN emp_cursor
    LOOP
        raise_salary(emp_rec.employee_id);
    END LOOP;
    COMMIT;
END process_emps;
/
```

# Handled Exceptions



```

CREATE PROCEDURE p2_ins_dept(p_locid NUMBER) IS
    v_did NUMBER(4);
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Procedure p2_ins_dept started');
    INSERT INTO departments VALUES (5, 'Dept 5', 145, p_locid);
    SELECT department_id INTO v_did FROM employees WHERE employee_id = 999;
END;
/

CREATE PROCEDURE p1_ins_loc(p_lid NUMBER, p_city VARCHAR2) IS
    v_city VARCHAR2(30); v_dname VARCHAR2(30);
BEGIN
    DBMS_OUTPUT.PUT_LINE('Main Procedure p1_ins_loc');
    INSERT INTO locations (location_id, city) VALUES (p_lid, p_city);
    SELECT city INTO v_city FROM locations WHERE location_id = p_lid;
    DBMS_OUTPUT.PUT_LINE('Inserted city '||v_city);
    DBMS_OUTPUT.PUT_LINE('Invoking the procedure p2_ins_dept ...');
    p2_ins_dept(p_lid);
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE ('No such dept / loc for any employee');
END;
/

```

The following code shows that the INSERT statements from both the procedures are successful:

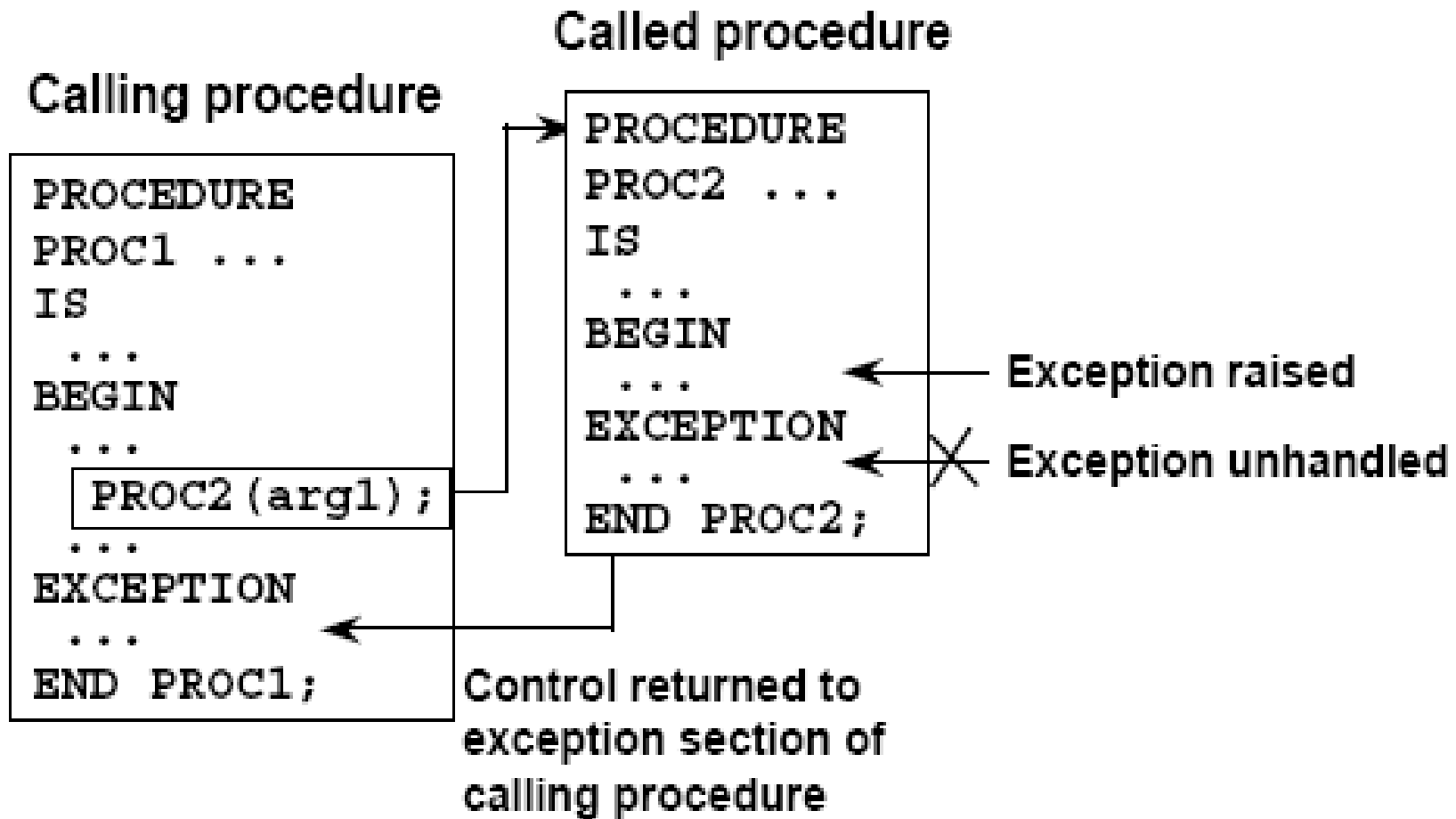
```
EXECUTE p1_ins_loc(1, 'Redwood Shores')  
SELECT location_id, city FROM locations  
WHERE location_id = 1;  
SELECT * FROM departments WHERE department_id = 5;
```

PL/SQL procedure successfully completed.

LOCATION_ID	CITY
1	Redwood Shores

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
5	Dept 5	145	1

# Unhandled Exceptions



```

CREATE PROCEDURE p2_noexcept(p_locid NUMBER) IS
    v_did NUMBER(4);
BEGIN
    DBMS_OUTPUT.PUT_LINE('Procedure p2_noexcept started');
    INSERT INTO departments VALUES (6, 'Dept 6', 145, p_locid);
    SELECT department_id INTO v_did FROM employees
    WHERE employee_id = 999;
END;
/

CREATE PROCEDURE p1_noexcept(p_lid NUMBER, p_city VARCHAR2) IS
    v_city VARCHAR2(30);
    v_dname VARCHAR2(30);
BEGIN
    DBMS_OUTPUT.PUT_LINE(' Main Procedure p1_noexcept');
    INSERT INTO locations (location_id, city) VALUES (p_lid, p_city);
    SELECT city INTO v_city FROM locations WHERE location_id = p_lid;
    DBMS_OUTPUT.PUT_LINE('Inserted new city '||v_city);
    DBMS_OUTPUT.PUT_LINE('Invoking the procedure p2_noexcept ...');
    p2_noexcept(p_lid);
END;
/

```



The following code shows that the DML statements from both the procedures are unsuccessful.

```
SET SERVEROUTPUT ON
EXECUTE p1_noexcep(3, 'New Delhi')
SELECT location_id, city FROM locations
WHERE location_id = 3;
SELECT * FROM departments WHERE department_id = 6;
```

```
Main Procedure p1_noexcep
Inserted new city New Delhi
Invoking the procedure p2_noexcep ...
Procedure p2_noexcep started
```

```
BEGIN p1_noexcep(3, 'New Delhi'); END;
```

```
*
```

```
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at "HR.P2_NOEXCEP", line 6
ORA-06512: at "HR.P1_NOEXCEP", line 10
ORA-06512: at line 1
```

# Removing Procedures

Drop a procedure stored in the database.

**Syntax:**

```
DROP PROCEDURE procedure_name
```

**Example:**

```
DROP PROCEDURE raise_salary;
```

Procedure dropped

# 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.**

# 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.**

# Practice 9 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 matching with 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**

