Chapter 6 Procedures

Named Program Blocks

- PL/SQL blocks executed thus far have been anonymous blocks
- Named PL/SQL block:
  - Procedures and Functions
  - Compiled and stored in the database as an object
  - Makes program units reusable

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

Anonymous Blocks and Subprograms

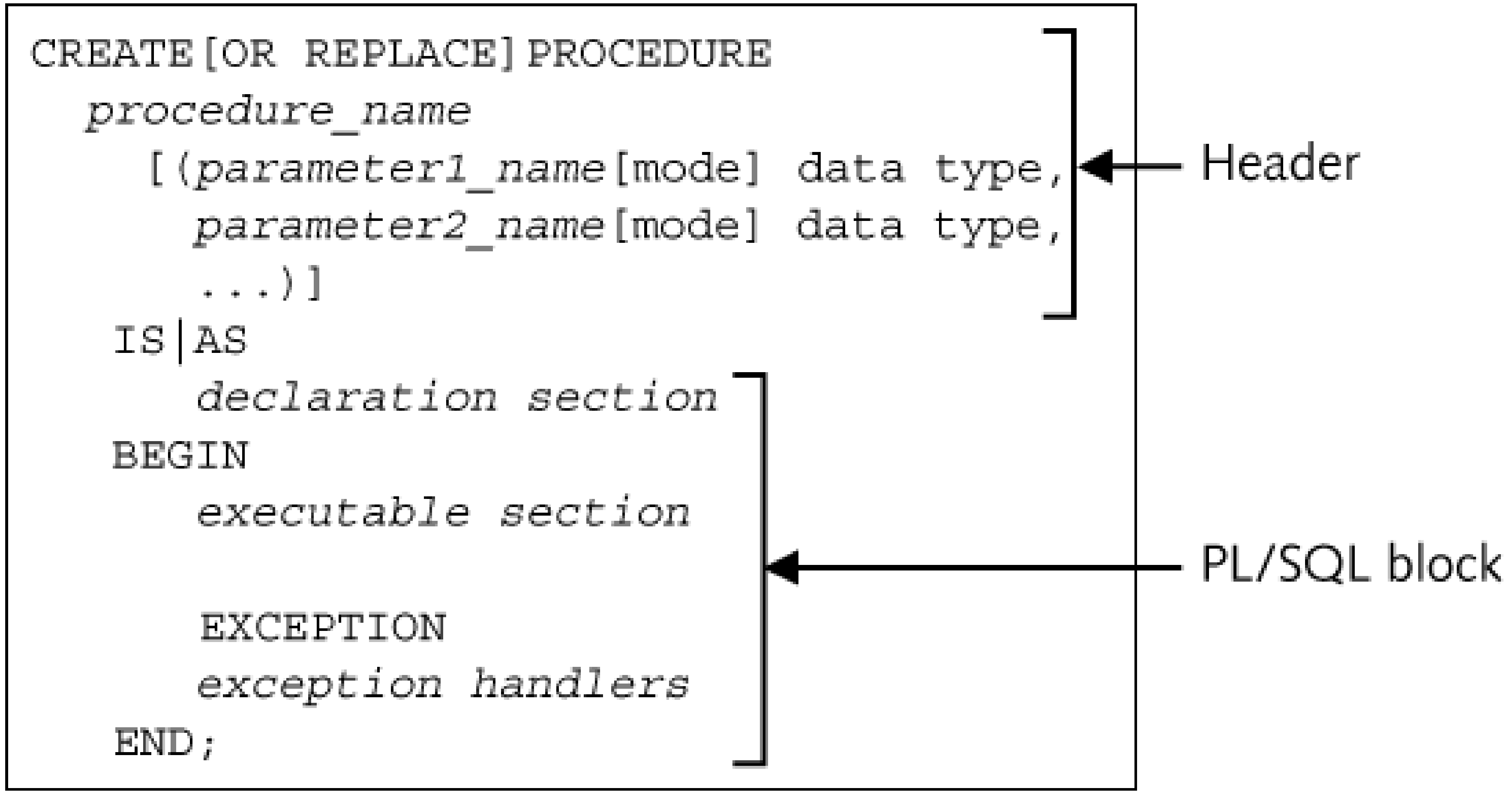
Anonymous Blocks

```
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)
```

CREATE PROCEDURE Syntax



```
CREATE [OR REPLACE] PROCEDURE
  procedure_name
    [(parameter1_name [mode] data type,
      parameter2_name [mode] data type,
      ...)]
   IS|AS
      declaration section
   BEGIN
      executable section

      EXCEPTION
      exception handlers
   END;
```

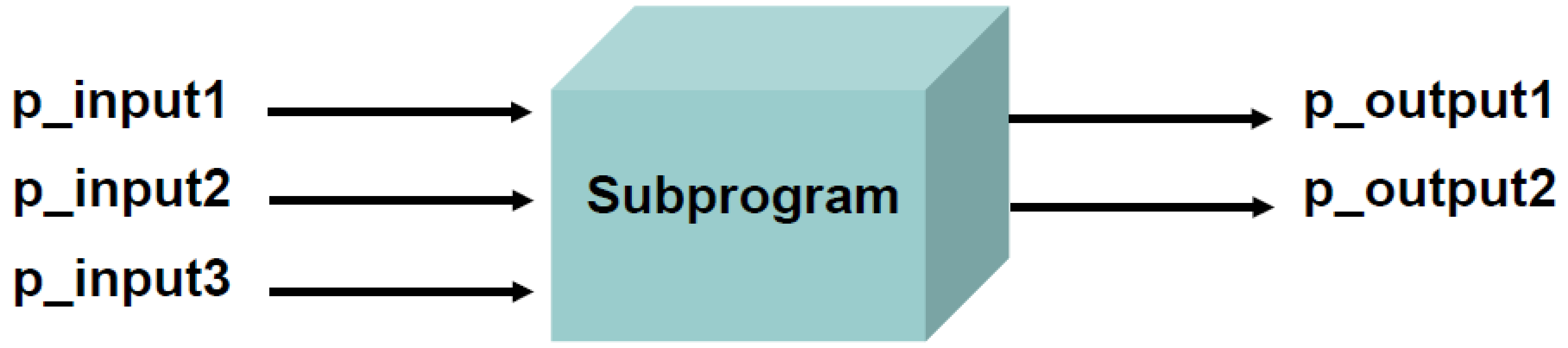
Invoking Procedures

- 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

Parameters

- Used to pass or communicate data between the caller and the subprogram
- Often named with a "p_" prefix
- Mode defaults to IN (next slide)



Parameters

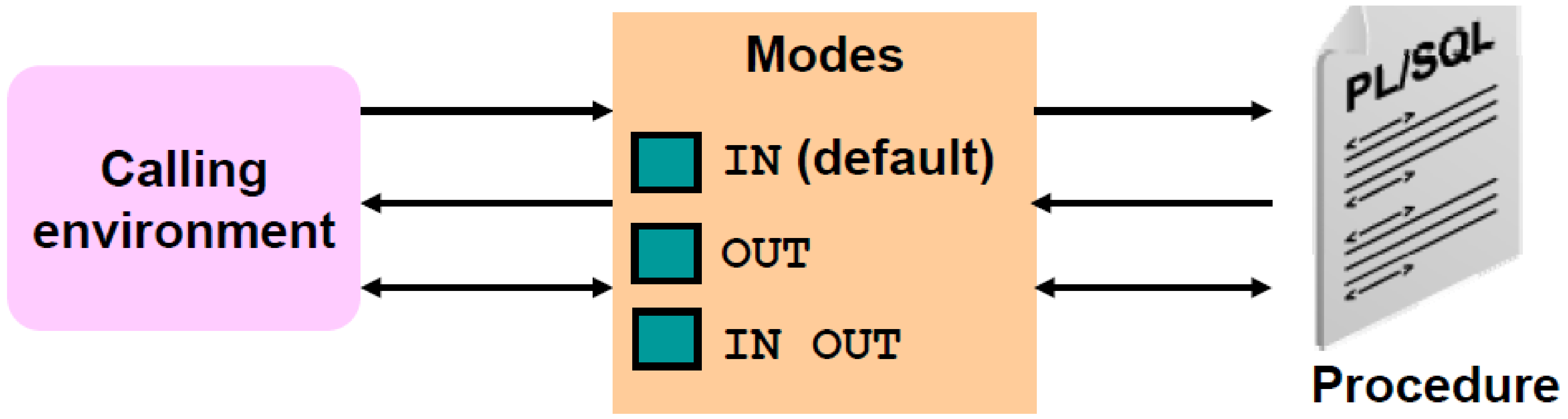
- Used to send values in and out of program units (subprograms)

MODE	DESCRIPTION
IN	Default if no mode is indicated. Passes a value from the application environment into the procedure. This value is considered a constant, as it cannot be changed within the procedure.
OUT	Passes a value out of the procedure to the application environment. If values are calculated or retrieved from the database within the procedure, OUT parameters are used to return these values to the calling environment.
IN OUT	Allows a value to be passed in and out using the same parameter. The values sent out can be different than the value sent in.

Arguments

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

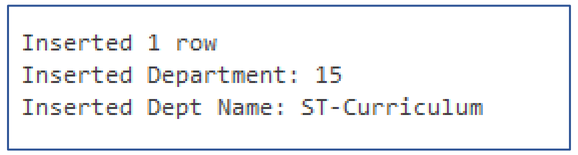
Parameters


Calling environment
Modes
IN (default)
OUT
IN OUT
PL/SQL
Procedure

```
/**********************************************************************
Example 1
-- Procedure with no parameters (parameters are optional)
-- Body is the same as an anonymous block
-- The declarative section of a procedure starts immediately after the procedure declaration (IS)
--                   and does not begin with the keyword DECLARE
***********************************************************************/
-- Create the DEPT table
DROP TABLE dept;

CREATE TABLE dept AS SELECT * FROM departments;

CREATE OR REPLACE PROCEDURE add_dept      -- Mandatory  -- Procedure name
IS                                        -- Mandatory – Followed by local variables and constants
  v_dept_id    dept.department_id%TYPE;   -- DECLARE not used -  Local variables and constants
  v_dept_name  dept.department_name%TYPE;
BEGIN                                     -- Mandatory
  v_dept_id   := 15;
  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');
  DBMS_OUTPUT.PUT_LINE('Inserted ' || 'Department: ' || v_dept_id );
  DBMS_OUTPUT.PUT_LINE('Inserted ' || 'Dept Name: '  || v_dept_name );
END;                                      -- Mandatory
```

```
Procedure created.
```

```
/***********************************************************************
Example 2
-- Invoke (execute) a procedure from an anonymous block
-- Can invoke (execute) a procedure from:
---- An anonymous block
---- Another procedure
---- A calling application
-- CANNOT invoke a procedure from inside a SQL statement such as SELECT
***********************************************************************/
SELECT * FROM dept;

-- Execute the ADD_DEPT procedure from an anonymous block
BEGIN
    add_dept;    -- Direct call to procedure
END;
```



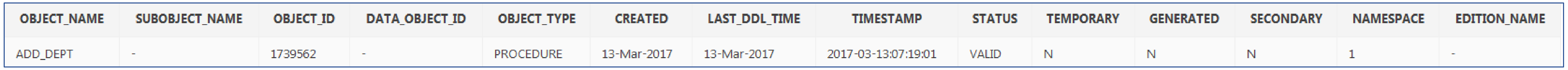
```
Inserted 1 row
Inserted Department: 15
Inserted Dept Name: ST-Curriculum
```

```
-- See the inserted row
SELECT *  FROM dept ORDER BY department_id;
```

```
DEPARTMENT_ID  DEPARTMENT_NAME  MANAGER_ID  LOCATION_ID
10             Administration          200         1700
15             ST-Curriculum             -            -
20             Marketing               201         1800
50             Shipping                124         1500
```

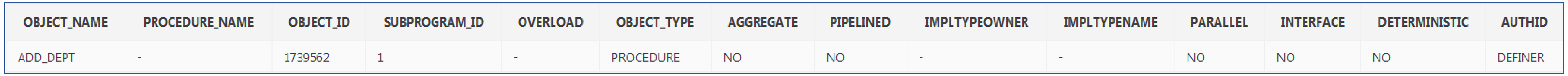
Obtaining information about procedures

```
-- Information about objects in USER_OBJECTS

SELECT *
    FROM USER_OBJECTS
    WHERE OBJECT_TYPE = 'PROCEDURE' AND object_name = 'ADD_DEPT';
```



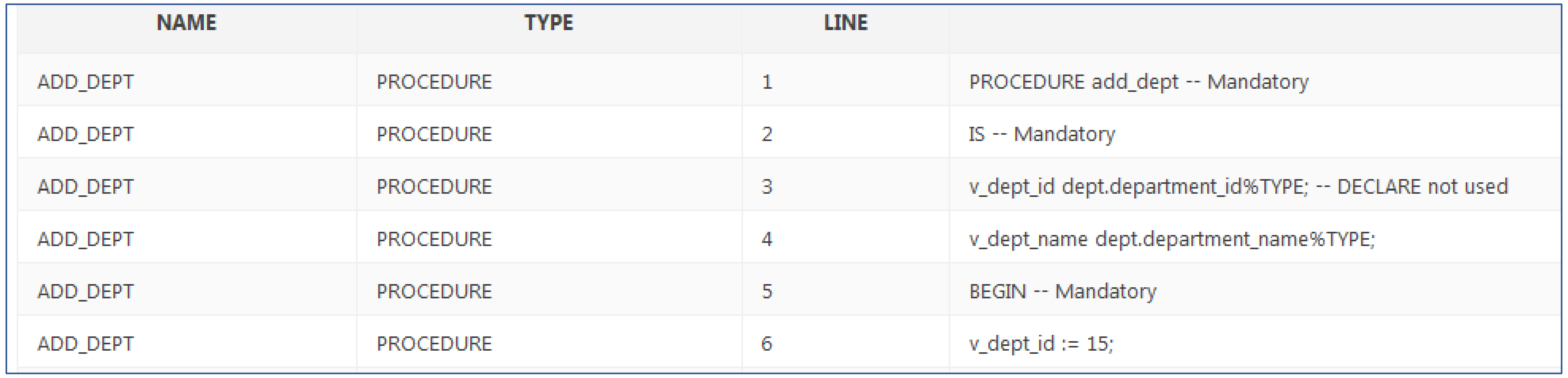
OBJECT_NAME	SUBOBJECT_NAME	OBJECT_ID	DATA_OBJECT_ID	OBJECT_TYPE	CREATED	LAST_DDL_TIME	TIMESTAMP	STATUS	TEMPORARY	GENERATED	SECONDARY	NAMESPACE	EDITION_NAME
ADD_DEPT	-	1739562	-	PROCEDURE	13-Mar-2017	13-Mar-2017	2017-03-13:07:19:01	VALID	N	N	N	1	-

```
-- Information about procedures in USER_PROCEDURES

SELECT *
    FROM USER_PROCEDURES
    WHERE OBJECT_TYPE = 'PROCEDURE' AND object_name = 'ADD_DEPT';
```



OBJECT_NAME	PROCEDURE_NAME	OBJECT_ID	SUBPROGRAM_ID	OVERLOAD	OBJECT_TYPE	AGGREGATE	PIPELINED	IMPLTYPEOWNER	IMPLTYPENAME	PARALLEL	INTERFACE	DETERMINISTIC	AUTHID
ADD_DEPT	-	1739562	1	-	PROCEDURE	NO	NO	-	-	NO	NO	NO	DEFINER

```
-- Information about source code in USER_SOURCE

SELECT *
    FROM USER_SOURCE
    WHERE NAME = 'ADD_DEPT';
```

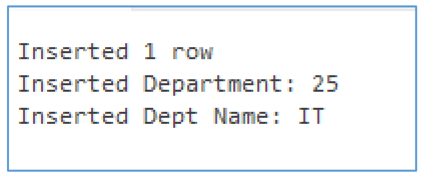


NAME	TYPE	LINE	
ADD_DEPT	PROCEDURE	1	PROCEDURE add_dept -- Mandatory
ADD_DEPT	PROCEDURE	2	IS -- Mandatory
ADD_DEPT	PROCEDURE	3	v_dept_id dept.department_id%TYPE; -- DECLARE not used
ADD_DEPT	PROCEDURE	4	v_dept_name dept.department_name%TYPE;
ADD_DEPT	PROCEDURE	5	BEGIN -- Mandatory
ADD_DEPT	PROCEDURE	6	v_dept_id := 15;

```
/**********************************************************************
Example 3
-- Parameters
---- Communicate data between the caller and the subprogram
---- Commonly referred to as arguments
------ Arguments are more appropriately thought of as the actual values assigned to the parameter variables
------     when the subprogram is called at runtime
---- MODE: IN, OUT, IN OUT
---- Prefix with p_

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

CREATE OR REPLACE PROCEDURE add_dept_with_parms
  (p_dept_id    IN   dept.department_id%TYPE,
   p_dept_name       dept.department_name%TYPE)  -- The IN mode is the default if no mode is specified
IS                                               -- No local variables defined between IS/BEGIN
BEGIN
  INSERT INTO dept(department_id, department_name)
    VALUES(p_dept_id, p_dept_name);
  DBMS_OUTPUT.PUT_LINE('Inserted '|| SQL%ROWCOUNT   ||' row');
  DBMS_OUTPUT.PUT_LINE('Inserted '|| 'Department: ' || p_dept_id );
  DBMS_OUTPUT.PUT_LINE('Inserted '|| 'Dept Name: '  || p_dept_name );
END;
```

```
Procedure created.
```

```
-- Invoke the procedure
BEGIN
  add_dept_with_parms(25,'IT');  -- Parameters are passed by positional notation by default
END;
```



```
Inserted 1 row
Inserted Department: 25
Inserted Dept Name: IT
```

```
SELECT * FROM dept ORDER BY department_id;

DEPARTMENT_ID   DEPARTMENT_NAME   MANAGER_ID   LOCATION_ID
           10   Administration           200          1700
           15   ST-Curriculum              -             -
           20   Marketing                201          1800
           25   IT                         -             -
           50   Shipping                 124          1500
```

```
DESCRIBE add_dept_with_parms;  -- Lists the parameters of a subprogram
```

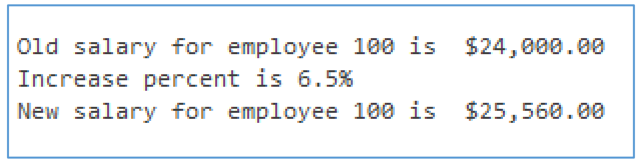
```
/*******************************************
Example 4
-- IN parameters
*******************************************/
DROP TABLE my_employees;

CREATE TABLE my_employees AS SELECT * FROM employees;

SELECT * FROM my_employees ORDER BY employee_id;



CREATE OR REPLACE PROCEDURE raise_salary
  (p_id       IN  my_employees.employee_id%TYPE,
   p_percent  IN  NUMBER)
IS
  v_old_salary my_employees.salary%TYPE;    -- Local variable
  v_new_salary my_employees.salary%TYPE;    -- Local variable
BEGIN
  SELECT salary INTO v_old_salary
    FROM my_employees
    WHERE employee_id = p_id;
  v_new_salary := v_old_salary * (1 + p_percent/100);
  UPDATE my_employees
    SET    salary = v_new_salary
    WHERE  employee_id = p_id;
  DBMS_OUTPUT.PUT_LINE('Old salary for employee ' || p_id || ' is ' || TO_CHAR( v_old_salary, '$99,999.99' ) );
  DBMS_OUTPUT.PUT_LINE('Increase percent is ' || p_percent || '%');
  DBMS_OUTPUT.PUT_LINE('New salary for employee ' || p_id || ' is ' || TO_CHAR( v_new_salary, '$99,999.99' ) );
  DBMS_OUTPUT.NEW_LINE();
END raise_salary;
```

```
Procedure created.
```

```
SELECT employee_id, salary FROM my_employees
  WHERE employee_id = 100;
```

```
EMPLOYEE_ID   SALARY
100            24000
```

```
-- Invoke the procedure
BEGIN
  raise_salary(100, 6.5);
END;
```



```
Old salary for employee 100 is  $24,000.00
Increase percent is 6.5%
New salary for employee 100 is  $25,560.00
```

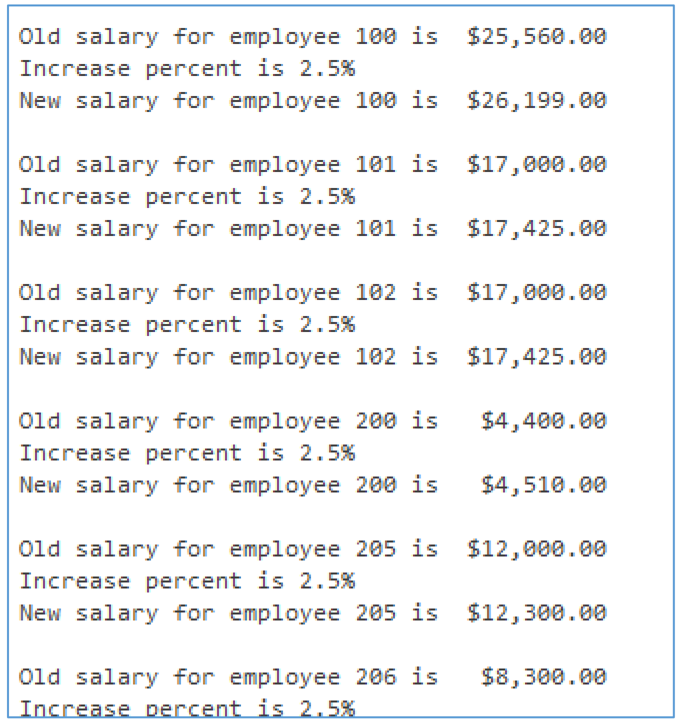
```
SELECT employee_id, salary FROM my_employees
  WHERE employee_id = 100;
```

```
EMPLOYEE_ID   SALARY
100            25560
```

```
/*******************************************
Example 5
-- Selecting multiple rows using a cursor
*******************************************/
CREATE OR REPLACE PROCEDURE process_employees
IS
  SALARY_INCREASE CONSTANT NUMBER (3,1) := 2.5;

  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, SALARY_INCREASE);  -- Invoke raise_salary procedure
   END LOOP;
   COMMIT;
END process_employees;
```

```
Procedure created.
```

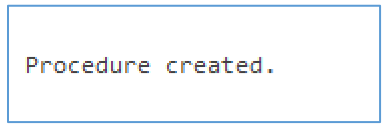
```
SELECT employee_id, last_name, salary
  FROM my_employees
  WHERE employee_id < 110;

EMPLOYEE_ID   LAST_NAME   SALARY
        100   King         25560
        101   Kochhar      17000
        102   De Haan      17000
        103   Hunold        9000
        104   Ernst         6000
        107   Lorentz       4200

-- Invoke the procedure
BEGIN
  process_employees;                          -- Invoke process_employees procedure which invokes raise_salary procedure
END;
```



```
Old salary for employee 100 is  $25,560.00
Increase percent is 2.5%
New salary for employee 100 is  $26,199.00

Old salary for employee 101 is  $17,000.00
Increase percent is 2.5%
New salary for employee 101 is  $17,425.00

Old salary for employee 102 is  $17,000.00
Increase percent is 2.5%
New salary for employee 102 is  $17,425.00

Old salary for employee 200 is   $4,400.00
Increase percent is 2.5%
New salary for employee 200 is   $4,510.00

Old salary for employee 205 is  $12,000.00
Increase percent is 2.5%
New salary for employee 205 is  $12,300.00

Old salary for employee 206 is   $8,300.00
Increase percent is 2.5%
```

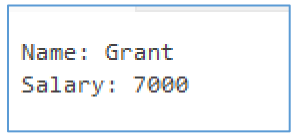
```
SELECT employee_id, last_name, salary
  FROM my_employees
  WHERE employee_id < 110;

EMPLOYEE_ID   LAST_NAME   SALARY
       100    King         26199
       101    Kochhar      17425
       102    De Haan      17425
       103    Hunold        9225
       104    Ernst         6150
       107    Lorentz       4305
```

```
/********************************************
Example 6
-- IN & OUT parameters
*******************************************/
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;
```



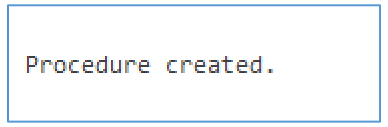
```
Procedure created.
```

```
-- Invoke the query_emp procedure from another procedure
-- Returns the employee name and salary
DECLARE
  v_emp_name    employees.last_name%TYPE;    -- Variable declared to hold value from OUT parameter
  v_emp_salary  employees.salary%TYPE;       -- Variable declared to hold value from OUT parameter
BEGIN
  query_emp(178, v_emp_name, v_emp_salary);  -- 1 IN parameter and 2 OUT parameters
  DBMS_OUTPUT.PUT_LINE('Name: ' || v_emp_name);
  DBMS_OUTPUT.PUT_LINE('Salary: ' || v_emp_salary);
END;
```



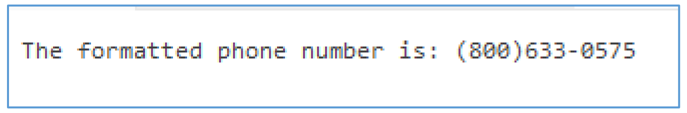
```
Name: Grant
Salary: 7000
```

```
/********************************************
Example 8
-- IN/OUT parameter
-- Send value IN and OUT via the same parameter
*******************************************/
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.
```

```
-- Invoke the format_phone procedure from another procedure
-- Returns formatted phone number
DECLARE
  v_phone_no VARCHAR2(13);
BEGIN
  v_phone_no := '8006330575' ;
  format_phone (v_phone_no);
  DBMS_OUTPUT.PUT_LINE('The formatted phone number is: ' || v_phone_no);
END;
```



```
The formatted phone number is: (800)633-0575
```

```
/*********************************************
Example 10
-- Parameters
---- Positional notation (Default)
---- Named (Keyword) notation
---- Combination (A positional parameter cannot follow a named parameter)
********************************************/

SELECT department_id, department_name, location_id
  FROM dept
  ORDER BY department_id;

-- Create the procedure
CREATE OR REPLACE PROCEDURE add_dept(
  p_dept_id   IN dept.department_id%TYPE,
  p_dept_name IN dept.department_name%TYPE,
  p_location  IN dept.location_id%TYPE)
IS
BEGIN
  INSERT INTO dept(department_id, department_name, location_id)
    VALUES (p_dept_id, p_dept_name, p_location);
END add_dept;
```

```
Procedure created.
```

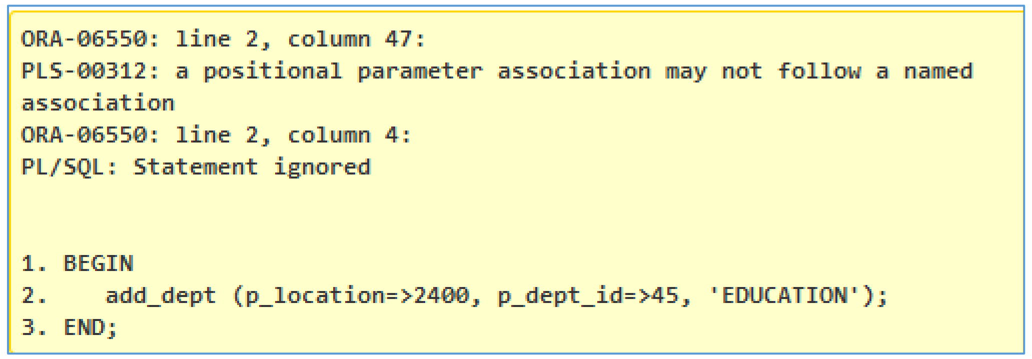
```
-- Invoke the procedure three times
-- Pass parameter by:
---- Positional notation
---- Named notation
---- Combination notation
-- A positional parameter cannot follow a named parameter

BEGIN
   add_dept (01, 'EDUCATION', 2100);                                         -- positional   - The values are specified by position
   add_dept (p_location=>2200, p_dept_name=>'EDUCATION', p_dept_id=>02); -- named        - The values are identified by parameter name
   add_dept (03, 'EDUCATION', p_location=>2300);                             -- combination  - A positional parameter cannot follow a named parameter
END;
```

```
Statement processed.
```

```
SELECT department_id, department_name, location_id
  FROM dept
  ORDER BY department_id;
```

```
DEPARTMENT_ID   DEPARTMENT_NAME   LOCATION_ID
            1   EDUCATION                2100
            2   EDUCATION                2200
            3   EDUCATION                2300
           10   Administration           1700
```

```
-- Invoke the procedure - will this work?
BEGIN
   add_dept (p_location=>2400, p_dept_id=>45, 'EDUCATION');
END;
```



```
ORA-06550: line 2, column 47:
PLS-00312: a positional parameter association may not follow a named
association
ORA-06550: line 2, column 4:
PL/SQL: Statement ignored


1. BEGIN
2.    add_dept (p_location=>2400, p_dept_id=>45, 'EDUCATION');
3. END;
```

```
/*******************************************
Example 11
-- Assign a default value for IN parameters
-- Two ways of assigning a default value to an IN parameter:
---- The assignment operator (:=), as shown for the p_dept_name parameter
---- The DEFAULT option, as shown for the p_location parameter

*******************************************/
SELECT department_id, department_name, location_id
  FROM dept
  ORDER BY department_id;

-- Create the procedure
CREATE OR REPLACE PROCEDURE add_dept(
  p_dept_id    IN  dept.department_id%TYPE,
  p_dept_name  IN  dept.department_name%TYPE :='Unknown',
  p_location   IN  dept.location_id%TYPE     DEFAULT 1100 )
IS
BEGIN
  INSERT INTO dept(department_id, department_name, location_id)
         VALUES (p_dept_id, p_dept_name, p_location);
END add_dept;
```

```
Procedure created.
```

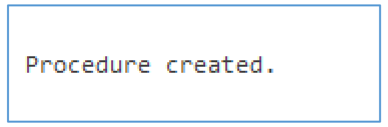
```
-- Invoke the procedure three times
BEGIN
  add_dept (05, 'ADVERTISING', p_location => 1200);
  add_dept (06);                              -- Uses default values p_dept_name and p_location
  add_dept (07, p_location => 1300);          -- Uses default value for p_name
  add_dept (08, 1400);                        -- Department name is 1400 and default value used for p_location
END;
```

```
Statement processed.
```

```
SELECT department_id, department_name, location_id
  FROM dept
  ORDER BY department_id;
```

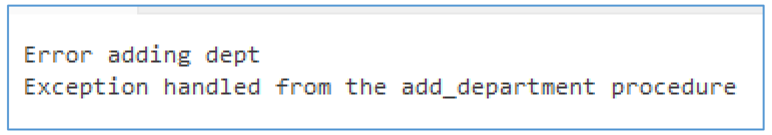
```
DEPARTMENT_ID   DEPARTMENT_NAME   LOCATION_ID
            5   ADVERTISING              1200
            6   Unknown                  1100
            7   Unknown                  1300
            8   1400                     1100
           10   Administration           1700
```

```
/**************************************************
Example 12 - Exception Handling between programs

**************************************************/
DROP TABLE departments_copy;
CREATE TABLE departments_copy AS SELECT * FROM departments;
ALTER TABLE departments_copy
  ADD CONSTRAINT departments_copy_pk
    PRIMARY KEY ( department_id );
ALTER TABLE departments_copy
  ADD CONSTRAINT departments_copy_fk
    FOREIGN KEY ( manager_id )
    REFERENCES employees( employee_id );

SELECT * FROM departments_copy;

-- Create the procedure
CREATE OR REPLACE PROCEDURE add_department(
  p_dept_id     IN  dept.department_id%TYPE,
  p_dept_name   IN  dept.department_name%TYPE,
  p_mgr_id      IN  dept.manager_id%TYPE,
  p_loc_id      IN  dept.location_id%TYPE )
IS
BEGIN
  INSERT INTO departments_copy(department_id, department_name, manager_id, location_id)
  VALUES (p_dept_id, p_dept_name, p_mgr_id, p_loc_id);
  DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_dept_name);
  EXCEPTION                                             -- Error is handled here (COULD ADD SPECIFIC ERROR NUMBERS)
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error adding dept');
      DBMS_OUTPUT.PUT_LINE('Exception handled from the add_department procedure');
END;
```



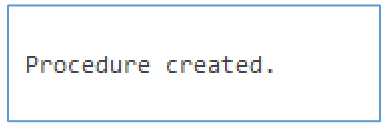
```
Procedure created.
```

```
-- Invoke the procedure with error
BEGIN
  add_department(06, 'Editing', 99, 1800); -- not a valid manager
END;
```



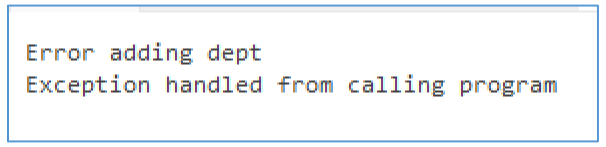
```
Error adding dept
Exception handled from the add_department procedure
```

```
/**************************************************
Example 13 - Exception Handling between programs
**************************************************/
-- Create the procedure
CREATE OR REPLACE PROCEDURE add_department_noex(
  p_dept_id    IN  dept.department_id%TYPE,
  p_dept_name  IN  dept.department_name%TYPE,
  p_mgr_id     IN  dept.manager_id%TYPE,
  p_loc_id     IN  dept.location_id%TYPE )
IS
BEGIN
  INSERT INTO departments(department_id, department_name, manager_id, location_id)
  VALUES (p_dept_id, p_dept_name, p_mgr_id, p_loc_id);
  DBMS_OUTPUT.PUT_LINE('Added Dept: '|| p_dept_name);        -- No EXCEPTION section
END;                                                          -- Exception not handled
                                                              -- Control returns to EXCEPTION section of calling program
```



```
Procedure created.
```

```
-- Invoke the procedure with error
BEGIN
  add_department_noex(02, 'Editing', 99, 1800);               -- not a valid mgr
EXCEPTION                                                     -- Error is returned to the EXCEPTION section of the calling program
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error adding dept');
    DBMS_OUTPUT.PUT_LINE('Exception handled from calling program');
END;
```



```
Error adding dept
Exception handled from calling program
```