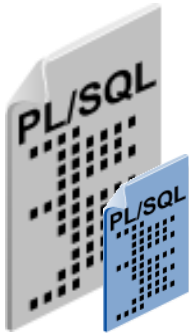
www.itsci.mju.ac.th/sayan

LEC 10: STORE PROCEDURE

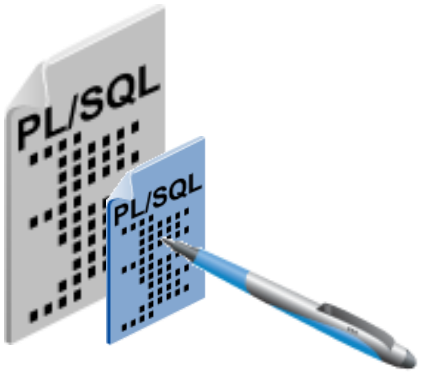
SAYAN UNANKARD
1/2558

WHAT ARE PL/SQL SUBPROGRAMS?

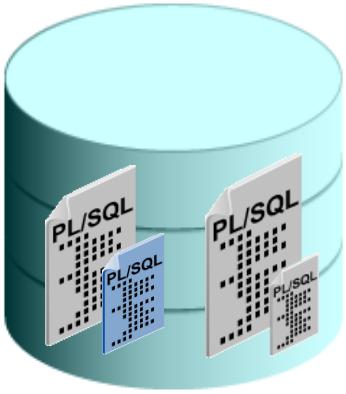
- PL/SQL โปรแกรมสามารถกำหนดชื่อเพื่อให้เรียกใช้ได้ และ สามารถส่งพารามิเตอร์เข้าไปยังโปรแกรมย่อย เหล่านั้นได้
- สามารถประกาศและพัฒนาโปรแกรมย่อยในส่วนของ PL/SQL block หรือ โปรแกรมย่อยอื่น ๆ ได้
- โปรแกรมย่อยจะต้องประกอบด้วยข้อกำหนดและส่วนประกอบหลัก
- สามารถกำหนดเป็น procedure หรือ function
- โดยปกติจะใช้ procedure เพื่อให้ทำงานใด ๆ และ function เพื่อประมวลผลและคืนค่าผลลัพธ์ที่ได้



THE BENEFITS OF USING PL/SQL SUBPROGRAMS



Easy maintenance



Subprograms: Stored procedures and functions

Improved data security and integrity

Improved code clarity

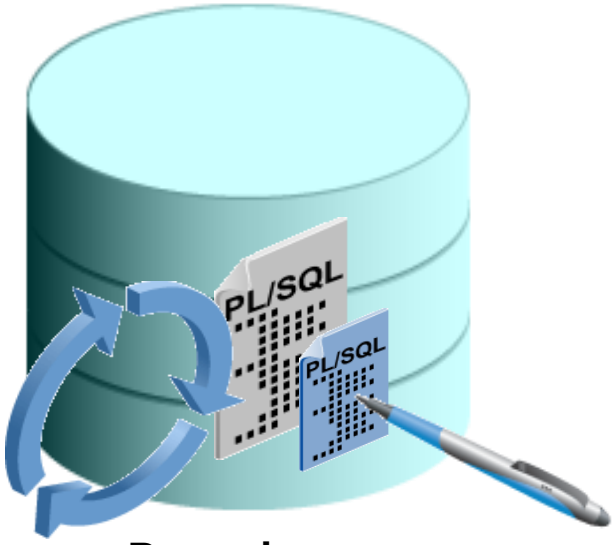
Improved performance

DIFFERENCES BETWEEN ANONYMOUS BLOCKS AND SUBPROGRAMS

Anonymous Blocks	Subprograms
ไม่มีชื่อเรียก PL/SQL blocks	มีการกำหนดชื่อของ PL/SQL blocks
คอมไพล์ทุกครั้ง	คอมไพล์เพียงครั้งเดียว
ไม่จัดเก็บในฐานข้อมูล	จัดเก็บในฐานข้อมูล
ไม่สามารถเรียกใช้ด้วยโปรแกรมอื่น ๆ	สามารถเรียกใช้ด้วยโปรแกรมอื่น ๆ
ไม่สามารถคืนค่าได้	ฟังก์ชันสามารถคืนค่าได้
ไม่สามารถกำหนดพารามิเตอร์ได้	สามารถกำหนดพารามิเตอร์ได้

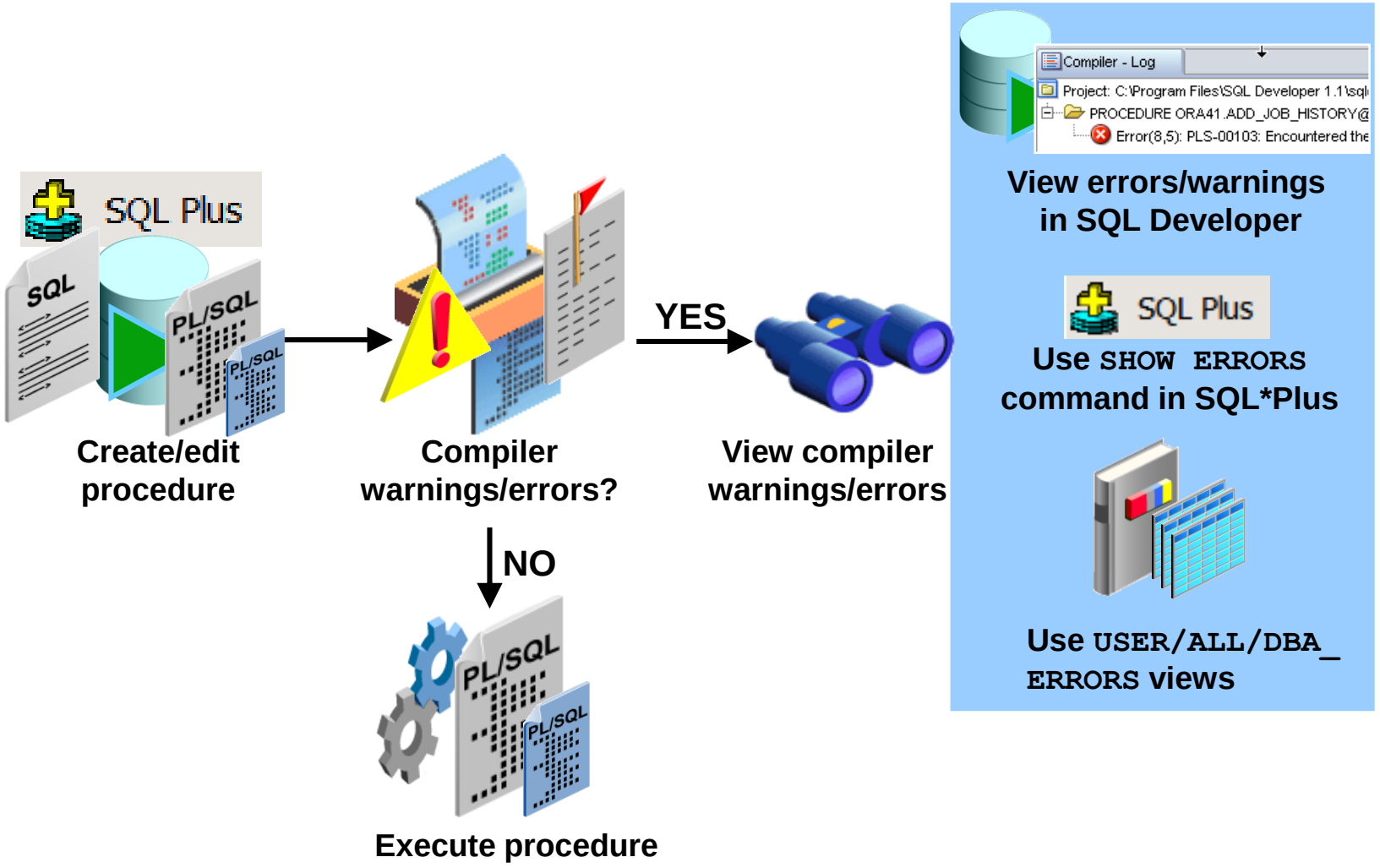
WHAT ARE PROCEDURES?

- เป็นชนิดหนึ่งของโปรแกรมย่อย ที่ใช้สำหรับการทำงานใด ๆ
- สามารถเก็บไว้ในฐานข้อมูลในรูปของ schema object
- สนับสนุนการนำกลับมาใช้ใหม่ และ การจัดการแก้ไขง่ายขึ้น



Procedures

CREATING PROCEDURES: OVERVIEW



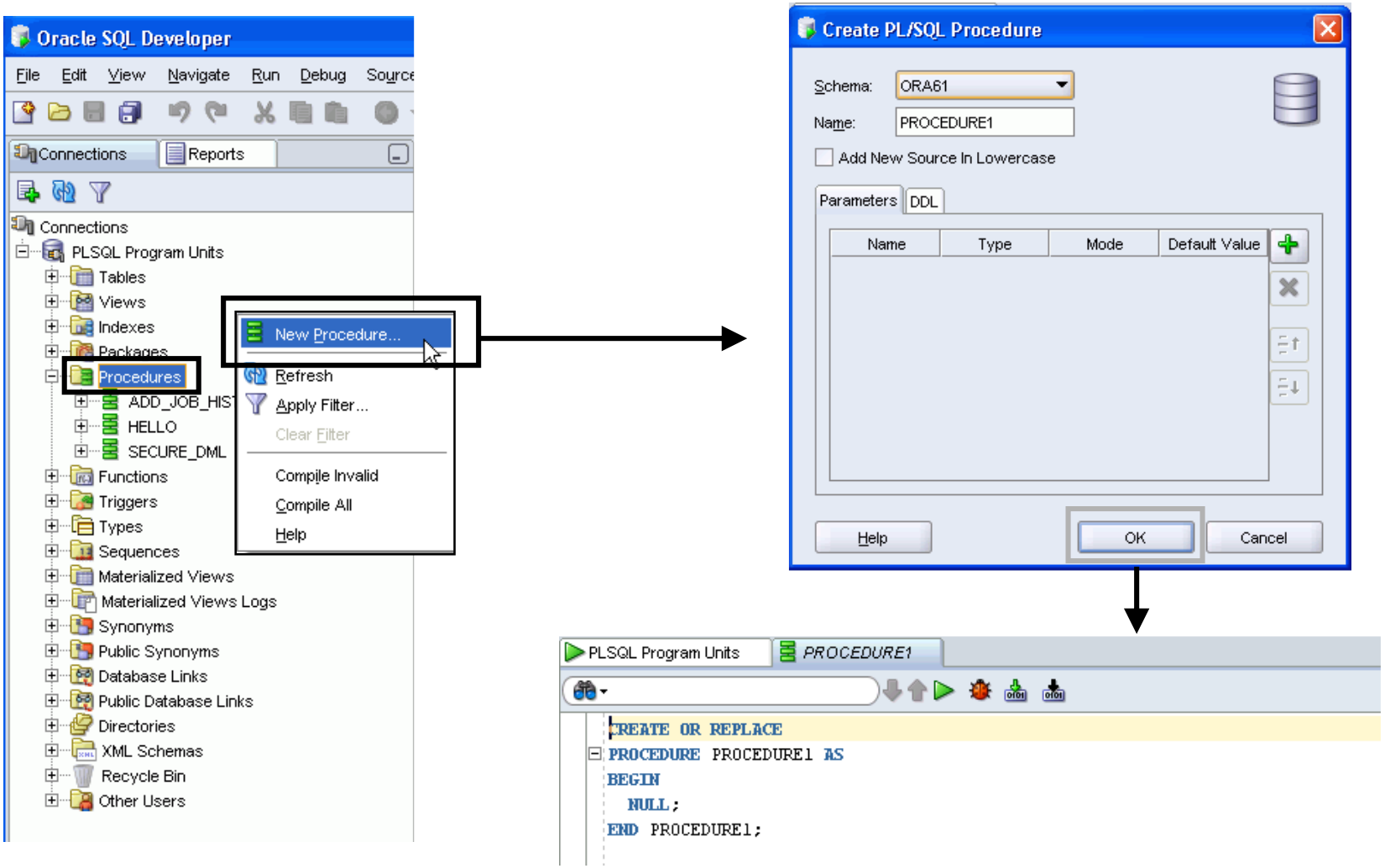
CREATING PROCEDURES

- ใช้คำสั่ง CREATE สำหรับการสร้างเพียงครั้งเดียวของ procedure ที่จะถูกจัดเก็บใน Oracle database
- ใช้คำสั่ง OR REPLACE กรณีที่ต้องการจะเขียนทับโปรแกรมเดิมที่มีอยู่

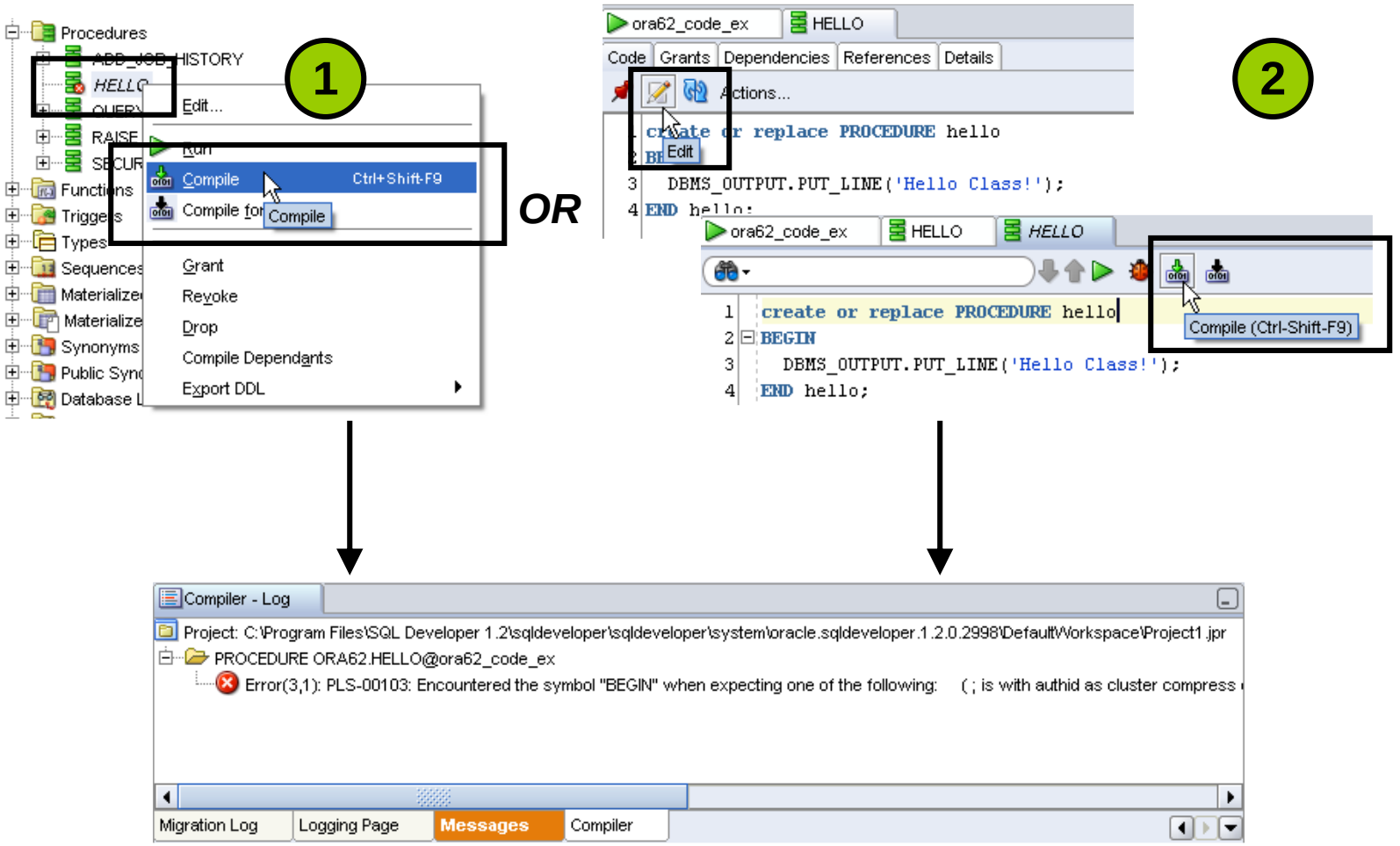
```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [(parameter1 [mode] datatype1,
   parameter2 [mode] datatype2, ...)]
IS|AS
  [local_variable_declarations; ...]
BEGIN
      -- actions;
END [procedure_name];
```

PL/SQL block

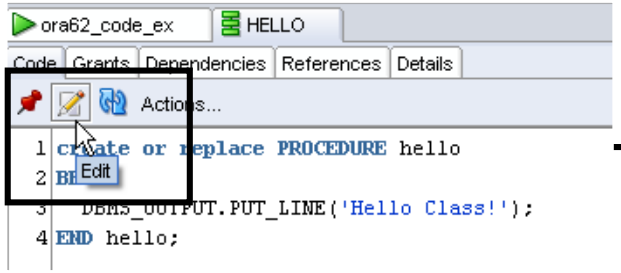
CREATING PROCEDURES USING SQL DEVELOPER



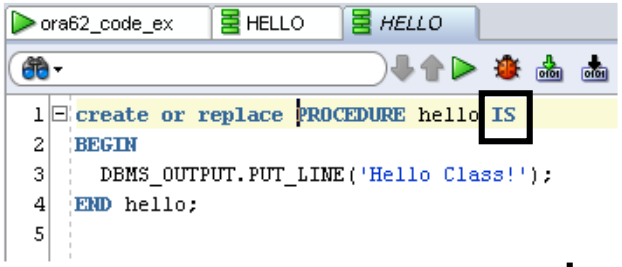
COMPILING PROCEDURES AND DISPLAYING COMPILATION ERRORS IN SQL DEVELOPER



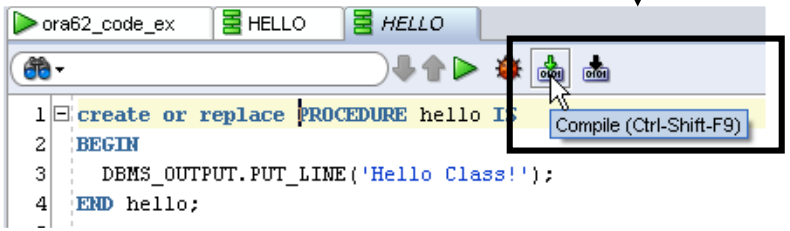
CORRECTING COMPILATION ERRORS IN SQL DEVELOPER



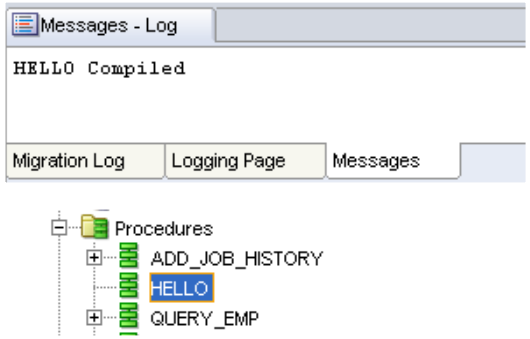
Edit procedure



Correct error



Recompile procedure



Recompilation successful

NAMING CONVENTIONS OF PL/SQL STRUCTURES USED IN THIS COURSE

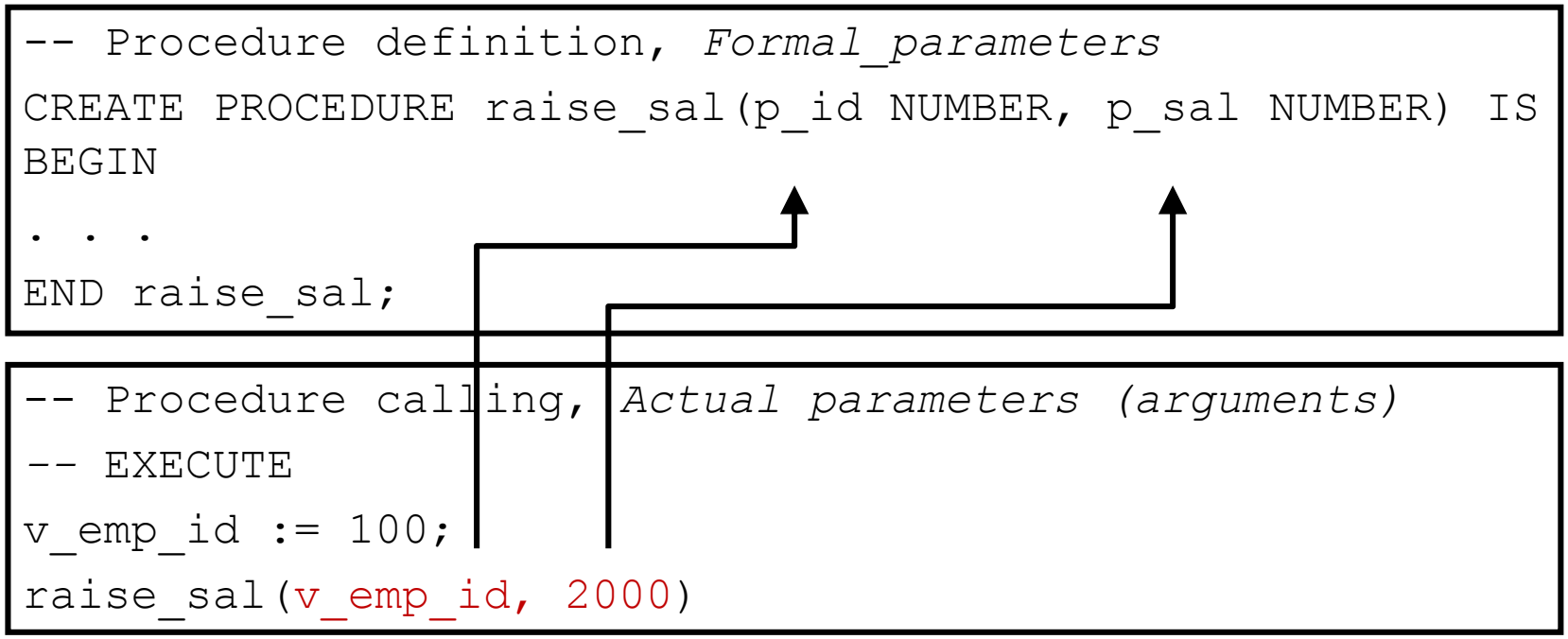
PL/SQL Structure	Convention	Example
Variable	v_*variable_name*	v_rate
Constant	c_*constant_name*	c_rate
Subprogram parameter	p_*parameter_name*	p_id
Bind (host) variable	b_*bind_name*	b_salary
Cursor	cur_*cursor_name*	cur_emp
Record	rec_*record_name*	rec_emp
Type	*type_name_type*	ename_table_type
Exception	e_*exception_name*	e_products_invalid
File handle	f_*file_handle_name*	f_file

WHAT ARE PARAMETERS AND PARAMETER MODES?

- พารามิเตอร์ถูกประกาศไว้หลังจากชื่อของโปรแกรมในส่วนของ PL/SQL header
- เป็นการส่งค่า หรือ สื่อสารข้อมูลระหว่างผู้เรียกกับโปรแกรมย่อย
- ใช้ในลักษณะเดียวกันกับตัวแปรประเภท local variables แต่ขึ้นอยู่กับประเภทของการส่งข้อมูล ซึ่งประกอบด้วย 3 แบบคือ
  - โหมด IN เป็นการส่งข้อมูลไปให้โปรแกรมย่อยอย่างเดียว
  - โหมด OUT เป็นพารามิเตอร์ที่ต้องการให้โปรแกรมย่อยคืนค่ากลับมายังผู้เรียกเพียงอย่างเดียว
  - โหมด IN OUT สามารถส่งข้อมูลไป และ คืนค่ากลับมาได้ทั้งสองทาง

FORMAL AND ACTUAL PARAMETERS

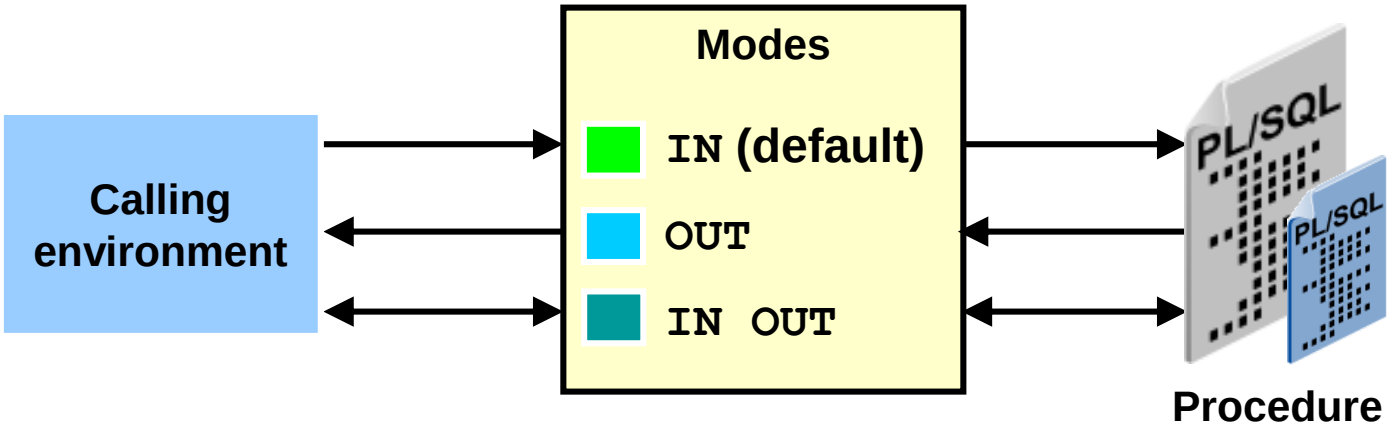
- Formal parameters: ใช้เรียกตัวแปรที่ประกาศไว้ในส่วนของหัวโปรแกรมย่อย
- Actual parameters (or arguments): ใช้เรียกตัวแปรที่เรียกใช้ในส่วนของโปรแกรมหลักที่เรียกใช้โปรแกรมย่อยนั้น ๆ



PROCEDURAL PARAMETER MODES

- โหมดของพารามิเตอร์จะถูกกำหนดในส่วนของ formal parameter declaration ระหว่างชื่อตัวแปร กับ ชนิดของตัวแปร
- โหมด IN เป็นโหมดตัวเลือกอัตโนมัติหากไม่มีการระบุโหมด

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



COMPARING THE PARAMETER MODES

IN	OUT	IN OUT
Default mode	จะต้องระบุ	จะต้องระบุ
สำหรับส่งค่าเข้าสู่โปรแกรมย่อย	สำหรับคืนค่าออกจากโปรแกรมย่อย	ทั้งส่งค่า และ คืนค่า
Formal parameter ทำหน้าที่คล้ายกับค่าคงที่	ไม่มีการกำหนดค่าไว้ก่อน	มีการกำหนดค่าก่อน
Actual parameter สามารถเป็นได้ทั้งตัวอักษร ข้อความ ค่าคงที่ ตัวแปรที่ระบุค่า	ต้องเป็นตัวแปรเท่านั้น	ต้องเป็นตัวแปรเท่านั้น
สามารถกำหนดตัวเลือกอัตโนมัติได้ (default)	ไม่สามารถกำหนดตัวเลือกอัตโนมัติได้	ไม่สามารถกำหนดตัวเลือกอัตโนมัติได้

USING THE IN PARAMETER MODE: EXAMPLE

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


```
EXECUTE raise_salary(176, 10)
```

USING THE OUT PARAMETER MODE: EXAMPLE

```
CREATE OR REPLACE PROCEDURE query_emp
 (p_id      IN  employees.employee_id%TYPE,
  p_name    OUT employees.last_name%TYPE,
  p_salary OUT employees.salary%TYPE) IS
BEGIN
  SELECT  last_name, salary INTO p_name, p_salary
  FROM    employees
  WHERE   employee_id = p_id;
END query_emp;
```

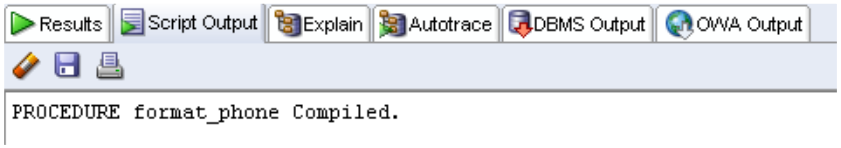
```
DECLARE
  v_emp_name employees.last_name%TYPE;
  v_emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(171, v_emp_name, v_emp_sal);
  DBMS_OUTPUT.PUT_LINE(v_emp_name||' earns '||
    to_char(v_emp_sal, '$999,999.00'));
END;/
```

USING THE IN OUT PARAMETER MODE: EXAMPLE

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;
/
```



USING THE IN OUT PARAMETER MODE: EXAMPLE

```
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;
/
```

```
DECLARE
   b_phone_no VARCHAR2(15):= '8006330575';
BEGIN
   DBMS_OUTPUT.PUT_LINE(b_phone_no);
   format_phone(b_phone_no);
   DBMS_OUTPUT.PUT_LINE(b_phone_no);
End;
```

VIEWING THE OUT PARAMETERS: USING THE DBMS_OUTPUT.PUT_LINE SUBROUTINE

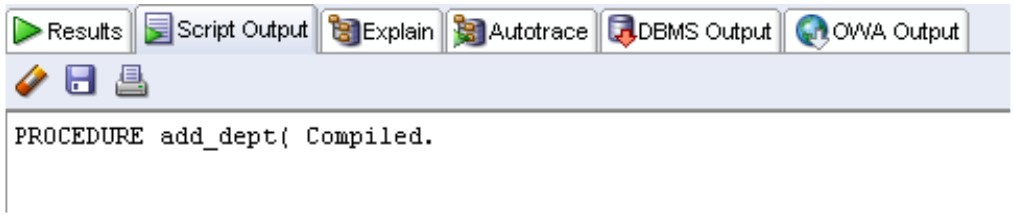
ใช้คำสั่ง PL/SQL สำหรับพิมพ์ค่าตัวแปรโดยการเรียกใช้โปรซีเยอร์ DBMS_OUTPUT.PUT_LINE

```
SET SERVEROUTPUT ON

DECLARE
  v_emp_name employees.last_name%TYPE;
  v_emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(171, v_emp_name, v_emp_sal);
  DBMS_OUTPUT.PUT_LINE('Name: ' || v_emp_name);
  DBMS_OUTPUT.PUT_LINE('Salary: ' || v_emp_sal);
END;
```

```
anonymous block completed
Name: Smith
Salary: 7400
```

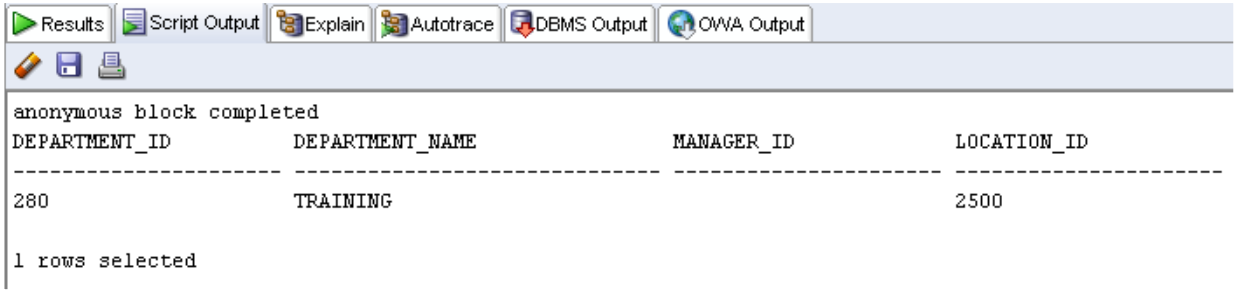
PASSING ACTUAL PARAMETERS: CREATING THE ADD_DEPT PROCEDURE

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

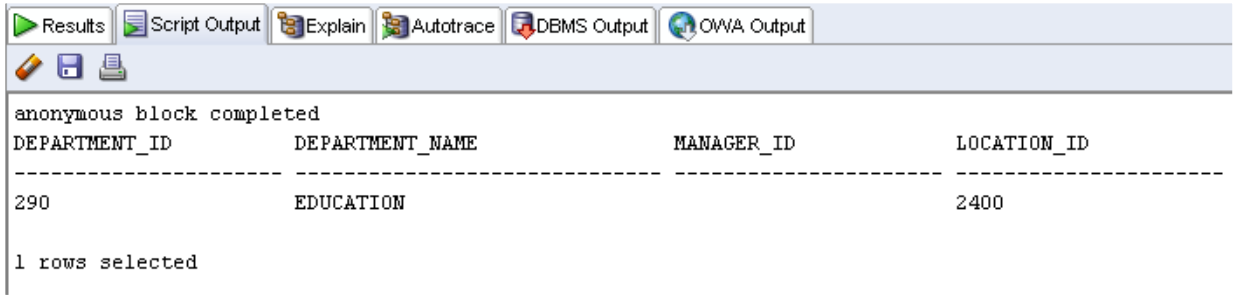


PASSING ACTUAL PARAMETERS: EXAMPLES

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




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



USING THE DEFAULT OPTION FOR THE PARAMETERS

- กำหนดตัวเลือกอัตโนมัติสำหรับพารามิเตอร์
- เพื่อความยืดหยุ่นโดยรวมเอาตำแหน่งและชื่อของพารามิเตอร์ที่ส่งไปอยู่ในโครงสร้าง

```
CREATE OR REPLACE PROCEDURE add_dept(
 p_name departments.department_name%TYPE:='Unknown',
 p_loc  departments.location_id%TYPE DEFAULT 1700)
IS
BEGIN
  INSERT INTO departments (department_id,
     department_name, location_id)
  VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_dept;
```

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

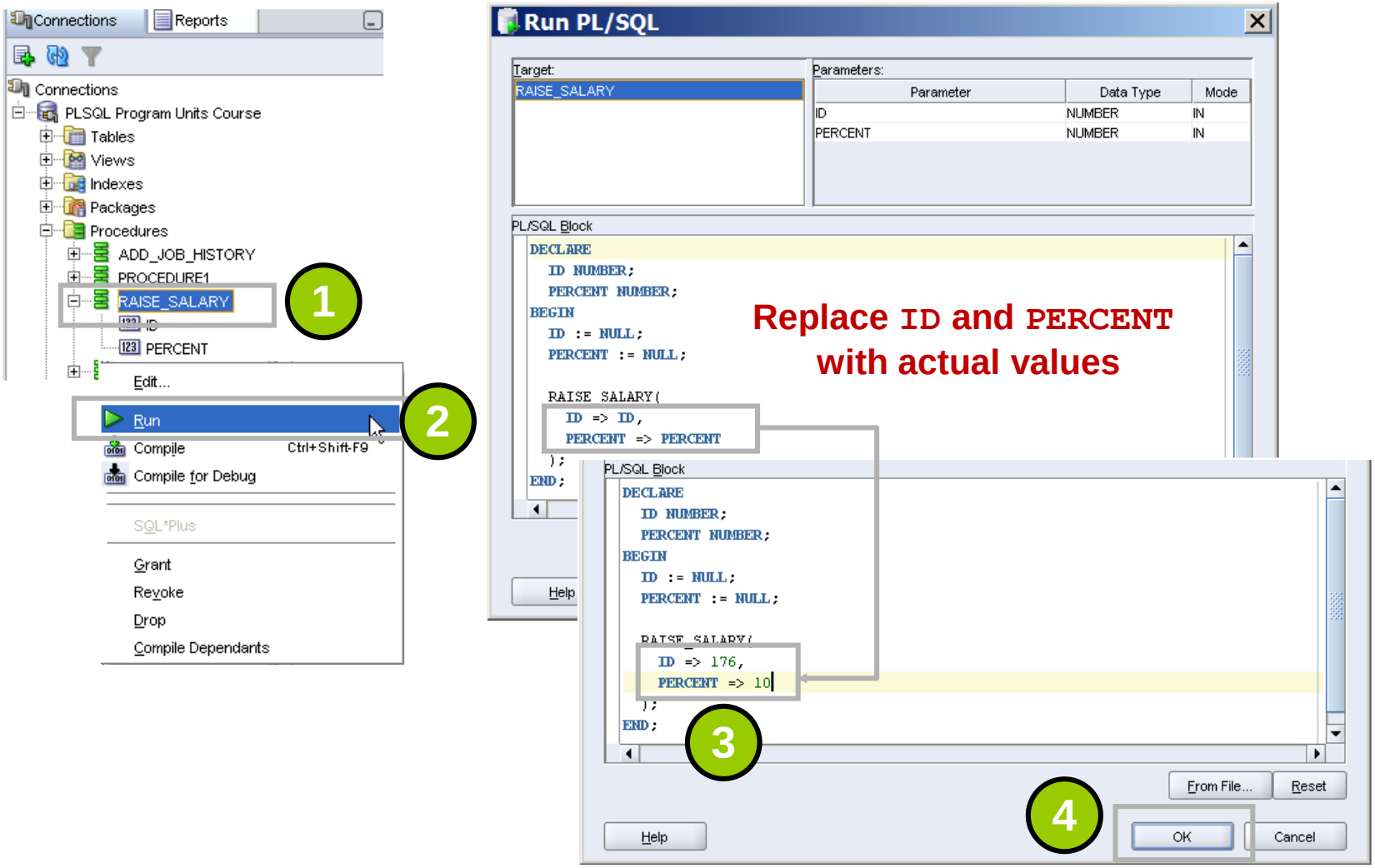
CALLING PROCEDURES

สามารถเรียกใช้โปรซีเยอร์ โดยใช้ anonymous blocks, procedure อื่น ๆ หรือ packages

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

```
PROCEDURE process_employees Compiled.
```

CALLING PROCEDURES USING SQL DEVELOPER

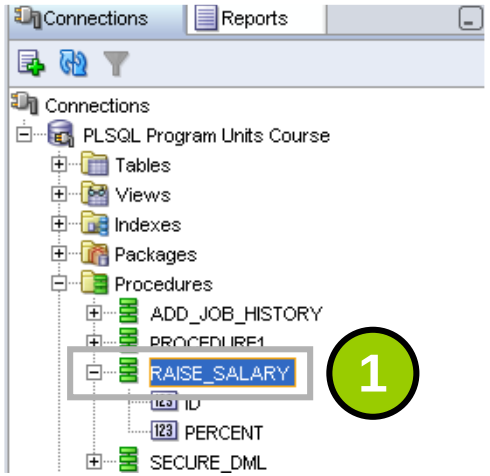


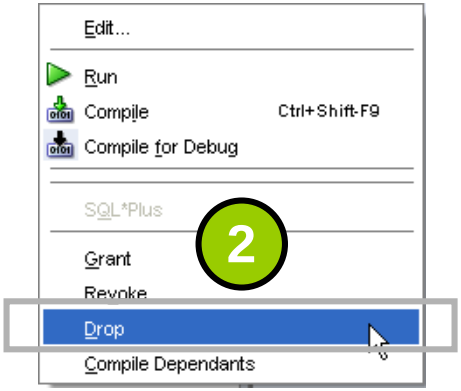
REMOVING PROCEDURES: USING THE DROP SQL STATEMENT OR SQL DEVELOPER

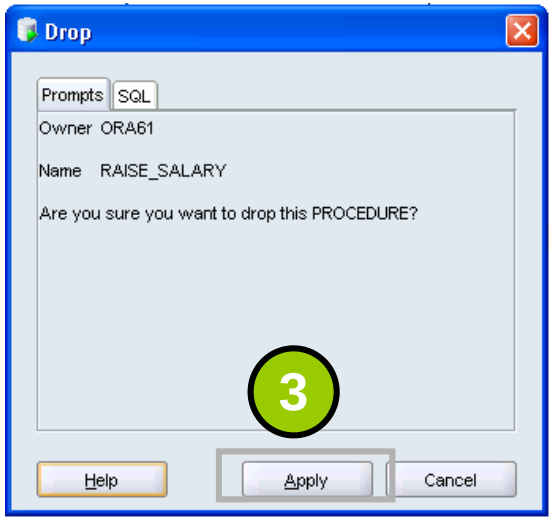
- ใช้คำสั่ง DROP ในการลบโปรซีเยอร์

```
DROP PROCEDURE raise_salary;
```

- การใช้ SQL Developer







VIEWING PROCEDURE INFORMATION USING THE DATA DICTIONARY VIEWS

DESCRIBE user_source --to display PL/SQL code that you own

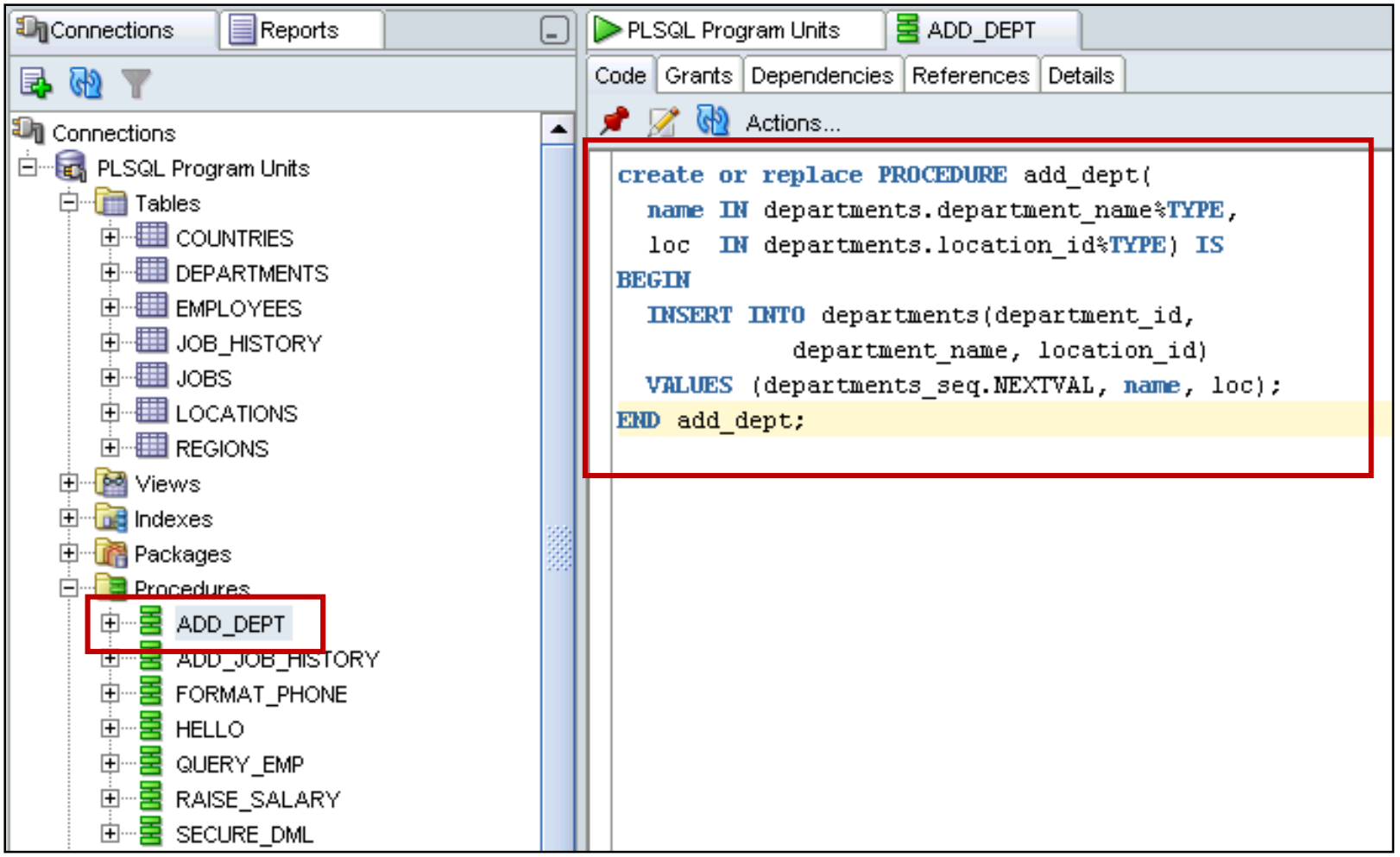
```
DESCRIBE user_source
Name                           Null     Type
------------------------------ -------- --------------------
NAME                                    VARCHAR2(30)
TYPE                                    VARCHAR2(12)
LINE                                    NUMBER
TEXT                                    VARCHAR2(4000)
```

```
SELECT text
FROM   user_source
WHERE  name = 'ADD_DEPT' AND type = 'PROCEDURE'
ORDER BY line;
```

Results:

	TEXT
1	PROCEDURE add_dept(
2	name IN departments.department_name%TYPE,
3	loc IN departments.location_id%TYPE) IS
4	BEGIN
5	INSERT INTO departments(department_id, department_name, location_id)
6	VALUES (departments_seq.NEXTVAL, name, loc);
7	END add_dept;

VIEWING PROCEDURE INFORMATION USING SQL DEVELOPER



การสร้างโปรแกรมให้เป็น STORED FUNCTION

OVERVIEW OF STORED FUNCTIONS

ฟังก์ชัน

- เป็นชื่อของชุดคำสั่ง PL/SQL block ที่สามารถคืนค่าได้
- สามารถจัดเก็บในฐานข้อมูลในรูปของ schema object เพื่อประมวลผลซ้ำได้
- มักจะถูกเรียกใช้ในคำสั่ง SQL หรือสำหรับการส่งผ่านพารามิเตอร์

CREATING FUNCTIONS

ในส่วนของ PL/SQL block จะต้องมีคำสั่งในการคืนค่าโดยใช้คำสั่ง RETURN

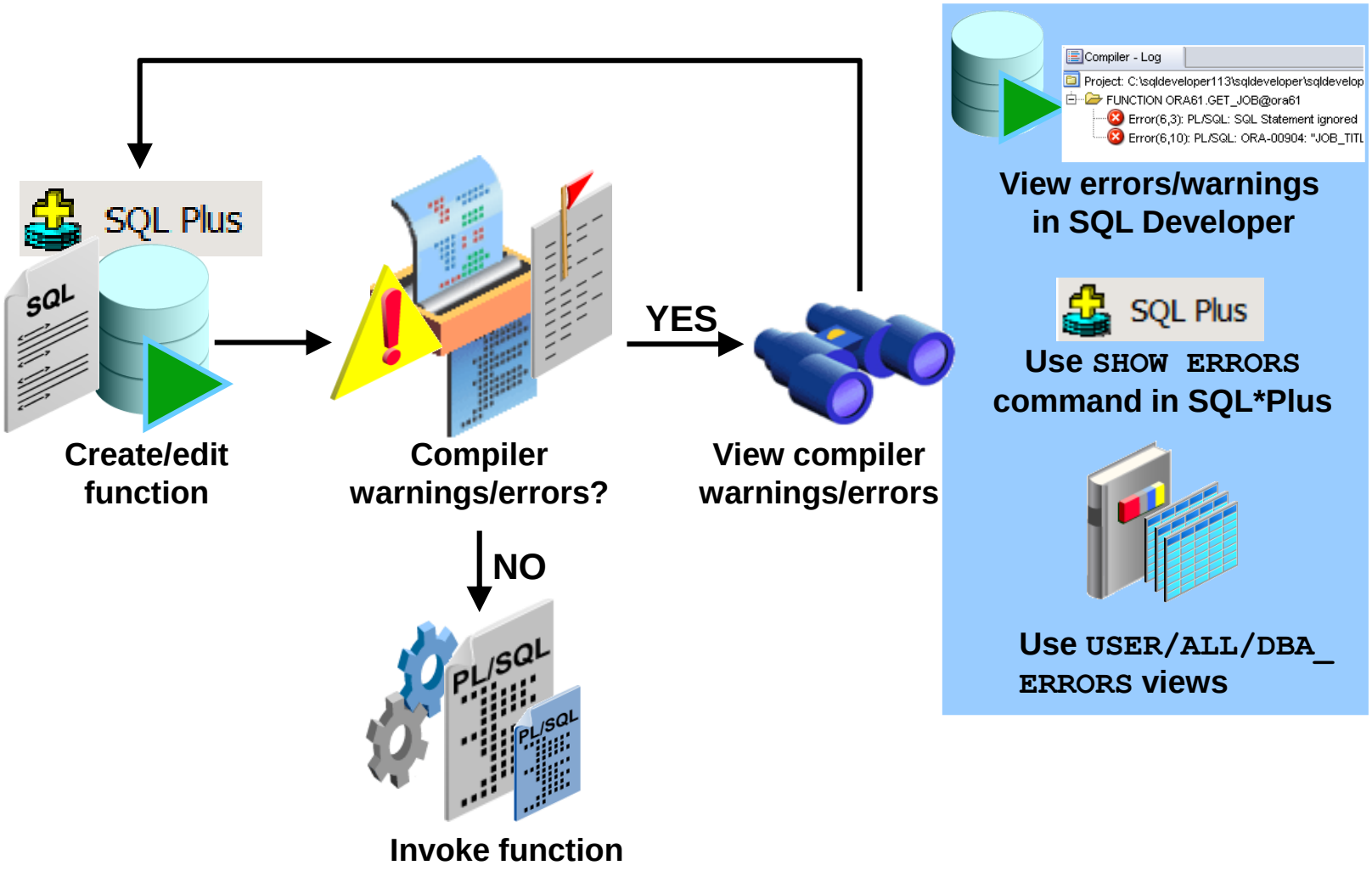
```
CREATE [OR REPLACE] FUNCTION function_name
  [(parameter1 [mode1] datatype1, . . .)]
RETURN datatype IS|AS
  [local_variable_declarations;
   . . .]
BEGIN
   -- actions;
   RETURN expression;
END [function_name];
```

PL/SQL Block

THE DIFFERENCE BETWEEN PROCEDURES AND FUNCTIONS

Procedures	Functions
เรียกใช้โดยคำสั่ง Execute	เป็นส่วนหนึ่งของคำสั่ง SQL
ไม่มีการคืนค่า	ต้องมีการคืนค่าในส่วนหัวของโปรแกรม
สามารถส่งค่าตัวแปรเข้าไปโดยใช้พารามิเตอร์	จะต้องคืนค่าเพียงค่าเดียว
สามารถประกอบด้วยคำสั่ง `RETURN` โดยไม่มีค่าข้อมูลได้	จะต้องคืนค่าด้วยคำสั่ง `RETURN` อย่างน้อย 1 ค่า

CREATING AND RUNNING FUNCTIONS: OVERVIEW



CREATING AND INVOKING A STORED FUNCTION USING THE CREATE FUNCTION STATEMENT: EXAMPLE

```
CREATE OR REPLACE FUNCTION get_sal
 (p_id employees.employee_id%TYPE) RETURN NUMBER IS
  v_sal employees.salary%TYPE := 0;
BEGIN
  SELECT salary
  INTO   v_sal
  FROM   employees
  WHERE  employee_id = p_id;
  RETURN v_sal;
END get_sal; /
```

```
FUNCTION get_sal Compiled.
```

```
-- Invoke the function as an expression or as
-- a parameter value.

EXECUTE dbms_output.put_line(get_sal(100))
```

```
anonymous block completed
24000
```

USING DIFFERENT METHODS FOR EXECUTING FUNCTIONS

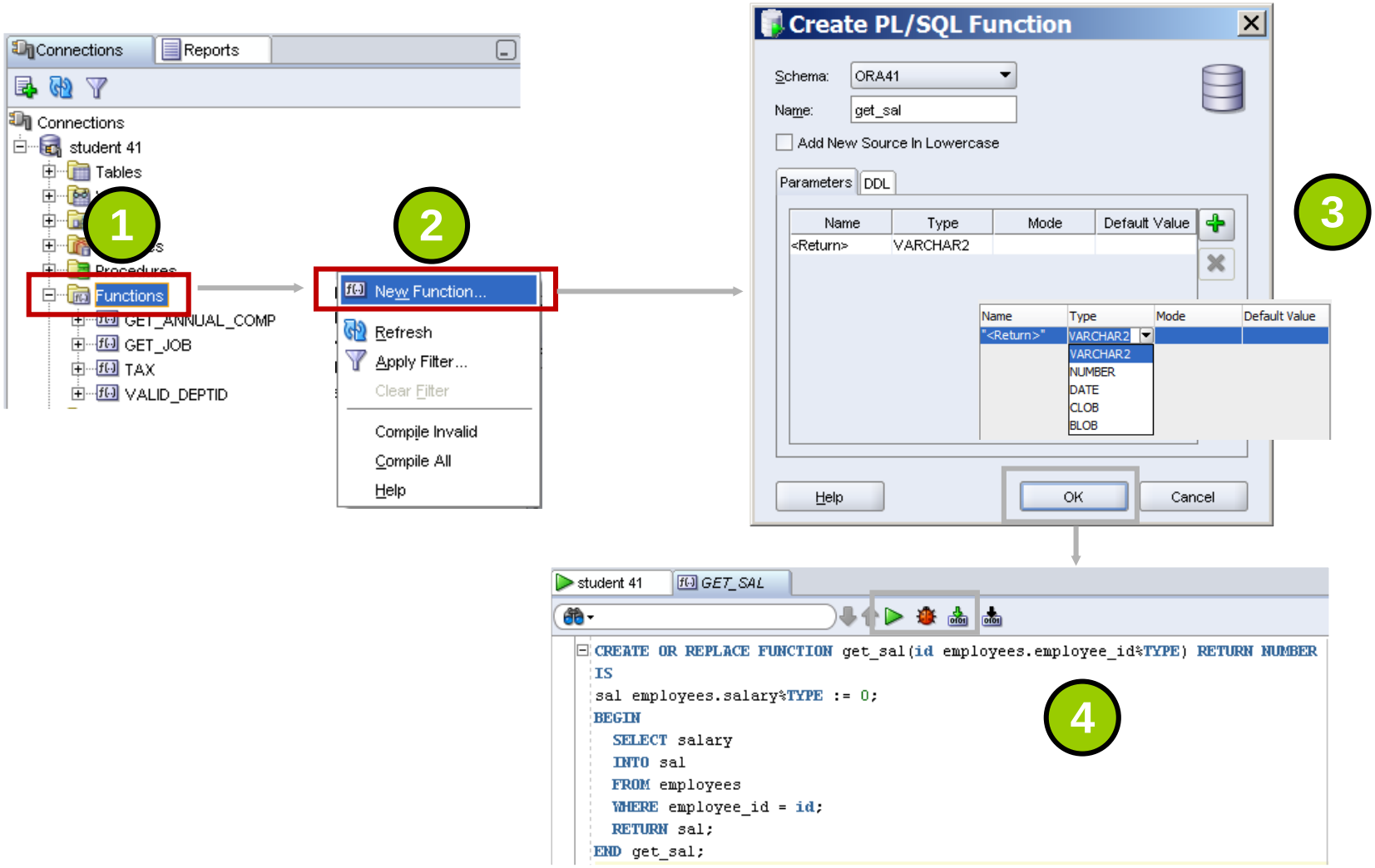
```
-- Use as a parameter to another subprogram

EXECUTE dbms_output.put_line(get_sal(100))
```

```
anonymous block completed
24000
```

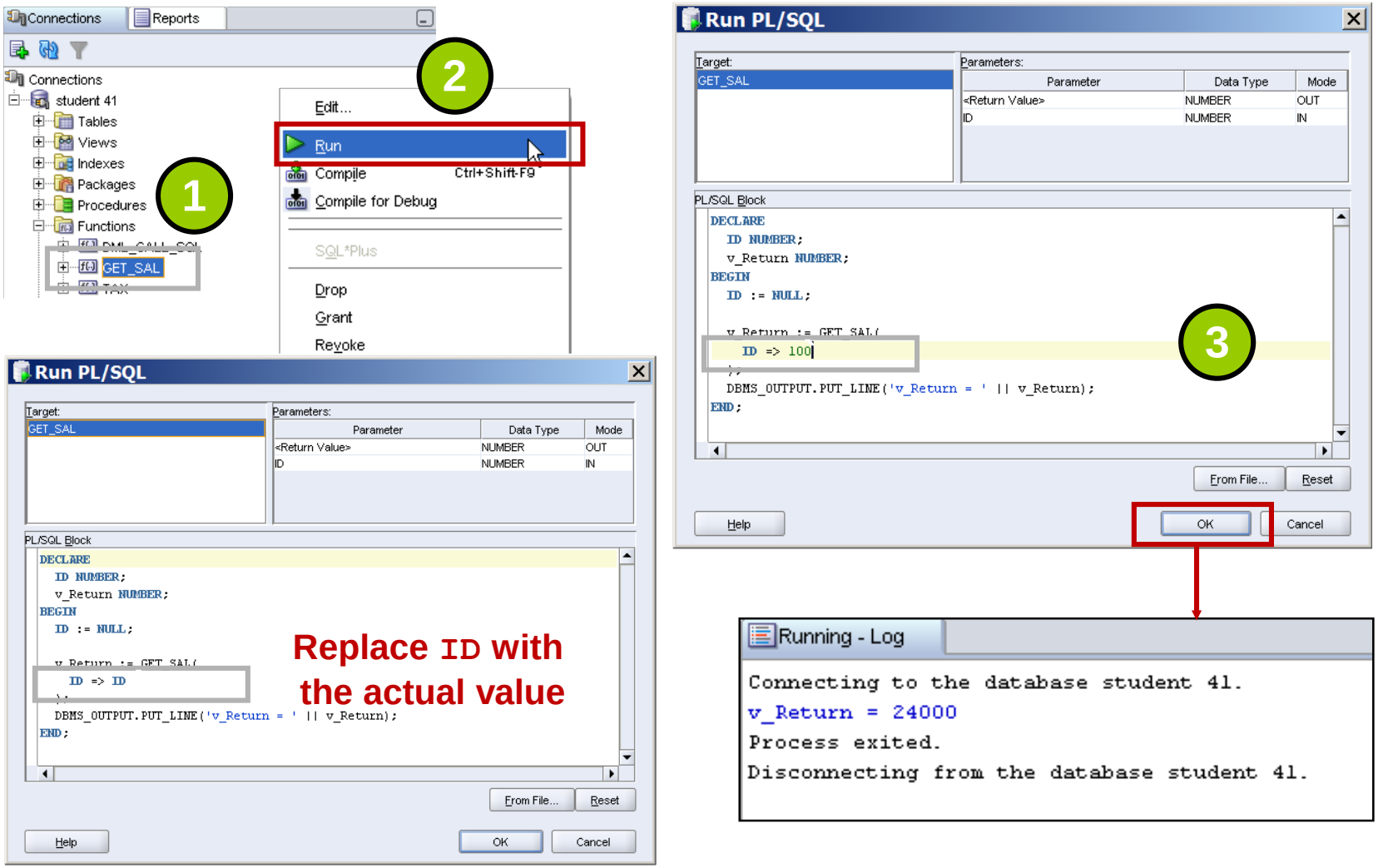
```
-- Use in a SQL statement (subject to restrictions)

SELECT job_id, get_sal(employee_id) FROM employees;
```

```
JOB_ID     GET_SAL(EMPLOYEE_ID)
---------- --------------------
SH_CLERK   2600
SH_CLERK   2600
AD_ASST    4400
MK_MAN     13000
```

CREATING AND COMPILING FUNCTIONS USING SQL DEVELOPER



EXECUTING FUNCTIONS USING SQL DEVELOPER

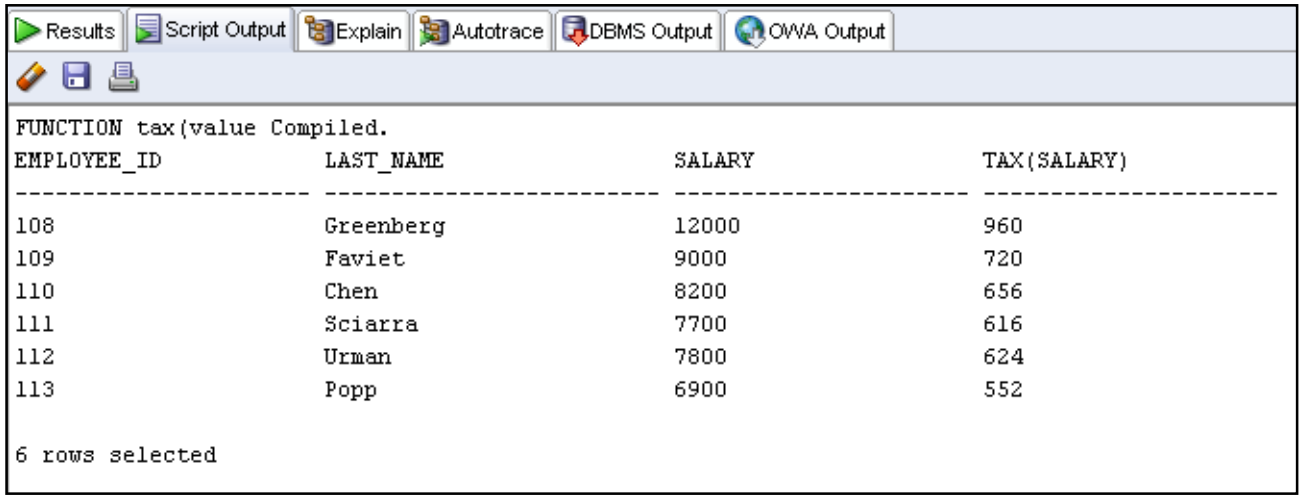


ADVANTAGES OF USER-DEFINED FUNCTIONS IN SQL STATEMENTS

- สามารถขยายคำสั่ง SQL ที่ซับซ้อน ไม่สะดวกสบาย หรือ ไม่สามารถใช้คำสั่ง SQL เพียงอย่างเดียวได้
- เพิ่มประสิทธิภาพมากขึ้นในการใช้คำสั่งในการกรองข้อมูลด้วยคำสั่ง WHERE และกรองข้อมูลในส่วนของแอปพลิเคชันด้วยเช่นกัน
- สามารถจัดการค่าของข้อมูลได้

USING A FUNCTION IN A SQL EXPRESSION: EXAMPLE

```
CREATE OR REPLACE FUNCTION tax(p_value IN NUMBER)
 RETURN NUMBER IS
BEGIN
   RETURN (p_value * 0.08);
END tax;
/
SELECT employee_id, last_name, salary, tax(salary)
FROM   employees
WHERE  department_id = 100;
```




```
FUNCTION tax(value Compiled.
EMPLOYEE_ID          LAST_NAME                 SALARY               TAX(SALARY)
-------------------- ------------------------- -------------------- --------------------
108                  Greenberg                 12000                960
109                  Faviet                    9000                 720
110                  Chen                      8200                 656
111                  Sciarra                   7700                 616
112                  Urman                     7800                 624
113                  Popp                      6900                 552

6 rows selected
```

RESTRICTIONS WHEN CALLING FUNCTIONS FROM SQL EXPRESSIONS

- ฟังก์ชันที่สร้างขึ้นโดยผู้ใช้ ซึ่งถูกเรียกจากคำสั่ง SQL จะต้อง
  - จัดเก็บในฐานข้อมูล
  - รองรับเฉพาะพารามิเตอร์โหมด IN ที่สนับสนุนชนิดข้อมูลของ SQL และต้องไม่ใช้ชนิดที่เจาะจงใน PL/SQL
  - คืนค่าได้เฉพาะชนิดข้อมูลของ SQL และต้องไม่ใช้ชนิดที่เจาะจงใน PL/SQL
- เมื่อมีการเรียกใช้ฟังก์ชันในคำสั่ง SQL
  - พารามิเตอร์จะต้องถูกกำหนดด้วยสัญลักษณ์ของตำแหน่งข้อมูล
  - จะต้องเป็นผู้สร้างฟังก์ชันหรือกำหนดสิทธิ์ให้ใช้งานได้เท่านั้น

CONTROLLING SIDE EFFECTS WHEN CALLING FUNCTIONS FROM SQL EXPRESSIONS

ฟังก์ชันถูกเรียกจาก

- คำสั่ง SELECT ไม่สามารถประกอบด้วยคำสั่ง DML

```
CREATE OR REPLACE FUNCTION DML_CALL_SQL (P_SALARY NUMBER)
RETURN NUMBER AS
BEGIN
  UPDATE employee SET SALARY = P_SALARY WHERE EMPID = 1;
  RETURN (P_SALARY + 100);
END DML_CALL_SQL;
```

```
select DML_CALL_SQL(SALARY) from employee WHERE EMPID = 1;
```

```
SQL Error: ORA-14551: cannot perform a DML operation inside a query
ORA-06512: at "XE.DML_CALL_SQL", line 3
14551. 00000 -  "cannot perform a DML operation inside a query "
*Cause:    DML operation like insert, update, delete or select-for-update
           cannot be performed inside a query or under a PDML slave.
*Action:   Ensure that the offending DML operation is not performed or
           use an autonomous transaction to perform the DML operation within
           the query or PDML slave.
```

CONTROLLING SIDE EFFECTS WHEN CALLING FUNCTIONS FROM SQL EXPRESSIONS

ฟังก์ชันถูกเรียกจาก

- คำสั่ง UPDATE หรือ DELETE กับตาราง T cannot query or contain DML on the same table T
- คำสั่ง SQL ไม่สามารถสิ้นสุดทรานแซคชัน ที่ประกอบด้วยคำสั่ง COMMIT หรือ ROLLBACK

RESTRICTIONS ON CALLING FUNCTIONS FROM SQL: EXAMPLE

```
CREATE OR REPLACE FUNCTION dml_call_sql(p_sal NUMBER)
   RETURN NUMBER IS
BEGIN
  INSERT INTO employees(employee_id, last_name,
                  email, hire_date, job_id, salary)
  VALUES(1, 'Frost', 'jfrost@company.com',
         SYSDATE, 'SA_MAN', p_sal);
  RETURN (p_sal + 100);
END;
```

```
UPDATE employees
  SET salary = dml_call_sql(2000)
WHERE employee_id = 170;
```

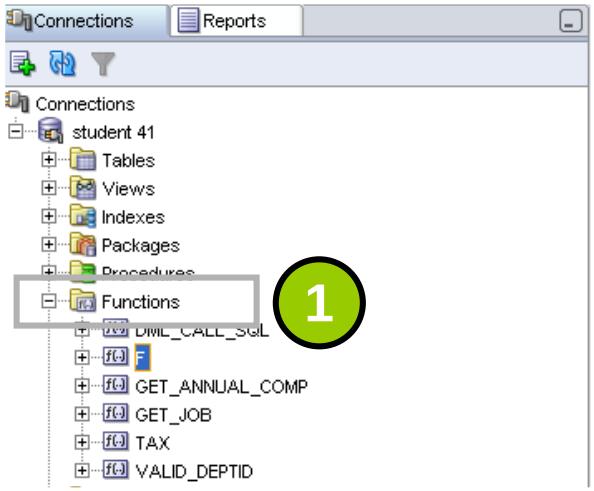
```
FUNCTION dml_call_sql(p_sal Compiled.

Error starting at line 1 in command:
UPDATE employees
  SET salary = dml_call_sql(2000)
WHERE employee_id = 170
Error report:
SQL Error: ORA-04091: table ORA62.EMPLOYEES is mutating, trigger/function may not see it
ORA-06512: at "ORA62.DML_CALL_SQL", line 4
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
*Cause:    A trigger (or a user defined plsql function that is referenced in
           this statement) attempted to look at (or modify) a table that was
           in the middle of being modified by the statement which fired it.
*Action:   Rewrite the trigger (or function) so it does not read that table.
```

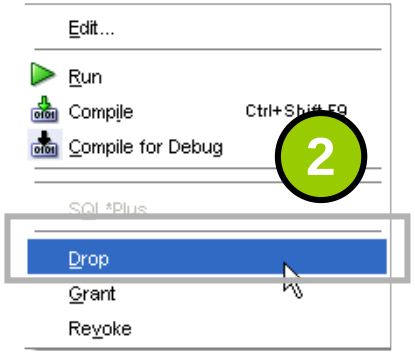
REMOVING FUNCTIONS: USING THE **DROP** SQL STATEMENT OR SQL DEVELOPER

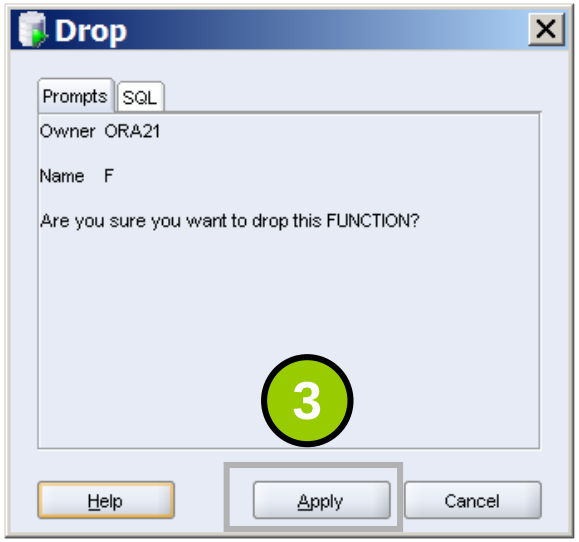
- การลบฟังก์ชันใช้คำสั่ง DROP

```
DROP FUNCTION f;
```

- การใช้ SQL Developer







VIEWING FUNCTIONS USING DATA DICTIONARY VIEWS

```
DESCRIBE USER_SOURCE
```

```
DESCRIBE user_source
Name                           Null     Type
------------------------------ -------- ----------------------------------------
NAME                                    VARCHAR2(30)
TYPE                                    VARCHAR2(12)
LINE                                    NUMBER
TEXT                                    VARCHAR2(4000)

4 rows selected
```

```
SELECT   text
FROM     user_source
WHERE    type = 'FUNCTION'
ORDER    BY line;
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	TEXT
1	FUNCTION get_sal
2	FUNCTION dml_call_sql(sal NUMBER)
3	FUNCTION tax(value IN NUMBER)
4	(id employees.employee_id%TYPE) RETURN NUMBER IS

...

19	FROM employees
20	END;
21	WHERE employee_id = id;
22	RETURN sal;
23	END get_sal;