PL/SQL

Procedures

DE HOGESCHOOL MET HET NETWERK

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

What Are PL/SQL Subprograms?

- A PL/SQL subprogram is a named PL/SQL block that can be called with a set of parameters.
- You can declare and define a subprogram within either a PL/SQL block or another subprogram.
- A subprogram consists of a specification and a body.
- A subprogram can be a procedure or a function.
- Typically, you use a procedure to perform an action and a function to compute and return a value.

Procedures and Functions

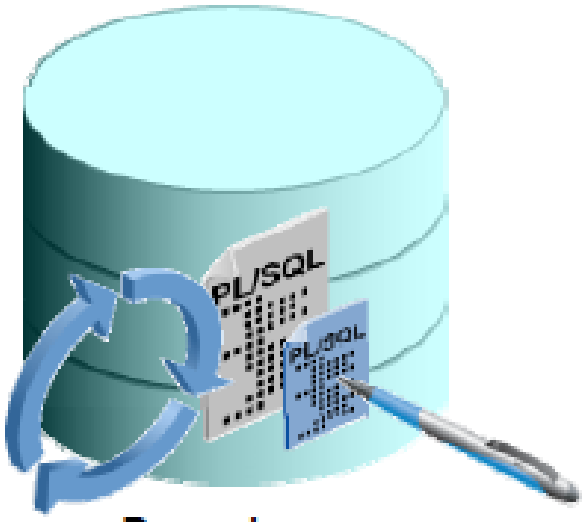
- Are named PL/SQL blocks
- Are called PL/SQL subprograms
- Have block structures similar to anonymous blocks:
  - Optional declarative section (without the `DECLARE` keyword)
  - Mandatory executable section
  - Optional section to handle exceptions

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 Are Procedures?

- Are a type of subprogram that perform an action
- Can be stored in the database as a schema object
- Promote reusability and maintainability



Procedures

Creating Procedures with the SQL CREATE OR REPLACE Statement

- Use the CREATE clause to create a stand-alone procedure that is stored in the Oracle database.
- Use the OR REPLACE option to overwrite an existing procedure.

```
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

Procedure – voorbeeld zonder parameters

```
CREATE OR REPLACE PROCEDURE show_emp
IS
        v_emp           employees.employee_id%type := '100';
        v_voornaam      employees.first_name%TYPE;
        v_naam          employees.last_name%TYPE;
BEGIN
        SELECT first_name, last_name
        INTO v_voornaam, v_naam
        FROM employees
        WHERE employee_id = v_emp;
        DBMS_OUTPUT.PUT_LINE(v_voornaam|| ' '|| v_naam);
END show_emp;
/
```

Let op: als employee_id 100 niet bestaat in de tabel employees dan zal de procedure afsluiten met de foutmelding: "no data found"

Procedure

- / → creatie procedure
- de broncode wordt in ieder geval in de data dictionary opgeslagen
- als foutloze code: gecompileerde versie → databank
- als code met fouten:
  `Melding: 'created with compilation errors'.`

  Hoe fouten opvragen: `show errors`

Procedure: Example

```
...
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;
```

Oproepen procedure

Vanuit een anoniem blok:

```
BEGIN
 add_dept;
END;
/
SELECT department_id, department_name FROM dept
WHERE department_id=280;
```

```
anonymous block completed
 Inserted 1 row

DEPARTMENT_ID          DEPARTMENT_NAME
---------------------- ---------------
280                    ST-Curriculum

1 rows selected
```

Vanuit SQL*Plus:

```
SQL> execute add_dept
```

OF exec add_dept

What Are Parameters and Parameter Modes?

- 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 mode (the default) provides values for a subprogram to process
  - An `OUT` parameter mode returns a value to the caller
  - An `IN OUT` parameter mode 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
- Actual parameters (or arguments): Literal values, variables, and expressions used in the parameter list of the calling subprogram

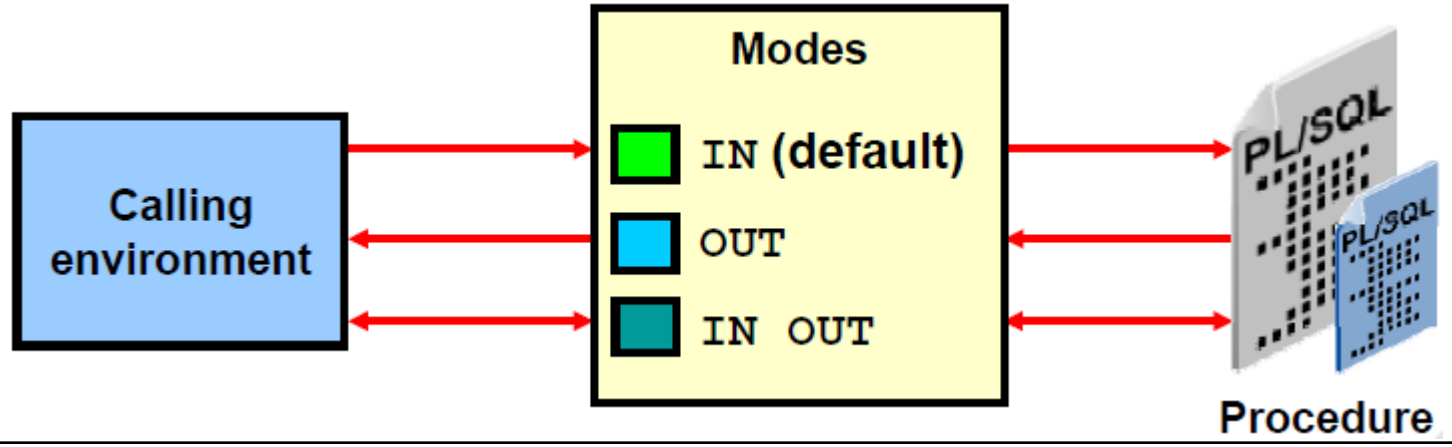
```
-- Procedure definition, Formal_parameters
CREATE PROCEDURE raise_sal(p_id NUMBER, p_sal NUMBER) IS
BEGIN
. . .
END raise_sal;
```

```
-- Procedure calling, Actual parameters (arguments)
v_emp_id := 100;
raise_sal(v_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 proc_name(param_name [mode] datatype)
...
```



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;
 /
```

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

```
PROCEDURE raise_salary Compiled.
```

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

Procedure met IN parameter(s)

```
CREATE OR REPLACE PROCEDURE del_ctry
        (p_country_id        IN        countries.country_id%TYPE)
IS
BEGIN
        DELETE FROM countries
        WHERE country_id = p_country_id;
        DBMS_OUTPUT.PUT_LINE('Er werden '|| SQL%ROWCOUNT||'
                        rijen verwijderd uit de tabel COUNTRIES');
END del_ctry;
/
```

Parameterlijst

```
(p_tekst        VARCHAR2,
 p_sal          employees.salary%TYPE,
 …)
```

naam	datatype
p_...	(geen lengte!!)

Meerdere parameters zijn gescheiden door een komma

Procedure met IN parameter(s)

- SQL> DESC[RIBE] del_ctry → geeft informatie over de parameters van de procedure del_ctry
- Procedure oproepen:

Vanuit een anoniem blok of een andere procedure:

in het BEGIN-blok: `del_ctry(v_country_id);`

OF `del_ctry('AR')`

Vanuit SQL*Plus:

```
SQL> exec del_ctry('AR')
```

OF `execute del_ctry('&landid')`

Procedure met IN en OUT parameter(s)

```
CREATE OR REPLACE PROCEDURE raise_salary_dept
        (p_dept_name IN departments.department_name%TYPE
        , p_percent  IN NUMBER
        , p_count_emp OUT NUMBER)
AS
        v_dept_id   departments.department_id%TYPE;
BEGIN
        SELECT department_id
        INTO v_dept_id
        FROM departments
        WHERE department_name = p_dept_name;
        UPDATE employees
        SET salary = salary * (1 + p_percent/100)
        WHERE department_id = v_dept_id;
        p_count_emp := SQL%ROWCOUNT;
END raise_salary_dept;
/
```

Procedure met IN en OUT parameter(s): andere oplossing

```
CREATE OR REPLACE PROCEDURE raise_salary_2_dept
        (p_dept_name IN departments.department_name%TYPE
        , p_percent      IN NUMBER
        , p_count_emp OUT NUMBER)
AS
BEGIN
        UPDATE employees
        SET salary = salary * (1 + p_percent/100)
        WHERE department_id = (SELECT department_id FROM departments
                                WHERE department_name = p_dept_name);
        p_count_emp := SQL%ROWCOUNT;
END raise_salary_2_dept;
/
```

Oproepen procedure

Vanuit een anoniem block:

```
DECLARE
        v_aantal_emp        NUMBER(3);
BEGIN
        raise_salary_dept('Administration', 10, v_aantal_emp);
        DBMS_OUTPUT.PUT_LINE(v_aantal_emp);
END;
/
```

Vanuit een andere procedure:

```
…
AS
        v_aantal_emp        NUMBER(3);
BEGIN
        raise_salary_dept('Administration', 10, v_aantal_emp);
        DBMS_OUTPUT.PUT_LINE(v_aantal_emp);
END;
/
```

- Vanuit het anonieme blok of de procedure wordt de naam van het departement nl. Administration en het percentage nl. 10 meegegeven aan het called program nl. raise_salary_dept
- Na het uitvoeren zal het aantal employees in het departement Administration met een loonsverhoging van 10% worden afgedrukt

Oproepen procedure

Vanuit SQL*Plus met een bind-variabele:

Bind variabele

1. deze variabele wordt gecreëerd in de werkomgeving en kan gebruikt worden in SQL statements en PL/SQL blocks
2. syntax aan SQL-prompt: VARIABLE b_test varchar2(2)
3. gebruikt als volgt :b_test

```
SQL> VARIABLE b_aantal_emp number
SQL> EXEC raise_salary_dept('Administration', 10, :b_aantal_emp)
SQL> PRINT b_aantal_emp
```

Om afdruk van bind variable automatisch te zien, voeg je de volgende setting toe in login.sql: SET AUTOPRINT ON

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;/
```

Procedure verwijderen

- Syntax:

```
DROP PROCEDURE procedure_name
```

- Voorbeeld: `DROP PROCEDURE raise_salary_dept;`
  - Alle privileges betreffende de procedure worden mee verwijderd.
  - De `CREATE OR REPLACE` syntax is equivalent aan het verwijderen en opnieuw creëren van de procedure. Toegekende privileges i.v.m. de procedure blijven bestaan als deze syntax gebruikt wordt.

Opvragen kenmerken (Data Dictionary)

Alle informatie over bestaande PL/SQL procedures en functies worden bewaard in de databank. Je kan hiervoor gebruik maken van volgende Oracle data dictionary views:

- `USER_OBJECTS`: deze view bevat informatie over ALLE databankobjecten van de eigen user, dus alle zelf-gecreëerde tabellen, indexen, sequences, functies, procedures,….
- `USER_SOURCE`: hierin zit de code van bepaalde objecten

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

SELECT text
FROM   user_source
WHERE  name = 'RAISE_SALARY_DEPT';
```