PL/SQL H3

Procedures

HOGESCHOOL PXL

Wat is een procedure?

- object (bestaat uit SQL-statements en PL/SQL-constructies) met een naam – ook subprogramma genoemd
- wordt bewaard in de DB
- code op 1 plaats definiëren en op meerdere plaatsen gebruiken
- voert één of meerdere acties uit
- kan aangeroepen(called) worden met 1 of meerdere parameters
- slechts 1 aanroep voor meerdere acties waardoor betere performance

Syntax voor de creatie van een procedure

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

Voorbeeld: procedure 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"

Afdrukken

- Gebruik maken van ingebouwde package DBMS_OUTPUT met mogelijke procedures o.a. PUT_LINE
- Tussen ronde haakjes de af te drukken lijn als 1 geheel meegeven
- Vb. DBMS_OUTPUT.PUT_LINE('Je kan tekst en variabelen samenvoegen met concatenatie-teken' || ' alle tekst moet tussen single quotes. Dit moet niet voor variabelen.' || v_country_name);
- in SQL*Plus SET SERVEROUTPUT ON opnemen → best in LOGIN.SQL
- LOGIN.SQL is een script dat telkens bij het opstarten van SQL*Plus wordt uitgevoerd en waar bepaalde settings kunnen opgenomen worden

/ → 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`

Voorbeeld: procedure zonder parameters

```
CREATE OR REPLACE PROCEDURE add_ctry
IS      v_country_id        countries.country_id%type := 'FR';
        v_country_name      countries.country_name%type := 'France';
        v_region_id         countries.region_id%type :=1;
BEGIN
        INSERT INTO countries
        VALUES (v_country_id,v_country_name,v_region_id);
        DBMS_OUTPUT.PUT_LINE('Er werden '|| SQL%ROWCOUNT||' rijen
        toegevoegd in de tabel COUNTRIES');
END add_ctry;
/
```

PL/SQL Syntax

- Elk DML-statement(INSERT, UPDATE, DELETE) kan zonder aanpassingen aan syntax opgenomen worden in PL/SQL
- Let op: een **DML-statement** in PL/SQL geeft geen fout als er 0 of meerdere rijen bewerkt worden
- Een **SELECT-statement** in PL/SQL dat 0 resultaatrijen geeft zal leiden tot de foutmelding ‘NO DATA FOUND’
- Een **SELECT-statement** in PL/SQL dat meerdere resultaatrijen geeft zal leiden tot de foutmelding ‘EXACT FETCH RETURNS MORE THAN REQUESTED NUMBER OF ROWS’

Impliciete cursor

Info omtrent het laatste SQL-statement staat in cursor **SQL**

- Cursor is een pointer naar een stukje geheugen
- Expliciete cursor: aangemaakt door gebruiker
- Impliciete cursor: aangemaakt door de Oracle Server o.a. **SQL**
- Toegang tot de info in de cursor kan via cursor attributen:

SQL%FOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement returned at least one row.
SQL%NOTFOUND	Boolean attribute that evaluates to TRUE if the most recent SQL statement did not return even one row.
SQL%ROWCOUNT	An integer value that represents number of rows affected by the most recent SQL statement.

Procedure oproepen

- Vanuit een andere procedure:

  in het BEGIN-blok: **`ADD_CTRY;`**

- Vanuit SQL*Plus:

```
SQL> execute add_ctry
```

`OF exec add_ctry`

Oefening 1

Oefening 2

Voorbeeld: 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;
/
```

PL/SQL Syntax - parameters

- Parameterlijst: zie hoofdstuk Functies

- IN-parameter komt binnen in de procedure (called program, subprogramma) en wordt meegegeven vanuit een andere procedure (calling program) of vanuit een aanroep in SQL*Plus

- De parameter(s) in het called program worden FORMAL-parameters genoemd

- De parameter(s) in het calling program worden ACTUAL-parameters genoemd

- Sql> DESC[RIBE] del_ctry — geeft informatie over de parameters van de procedure del_ctry

Procedure oproepen

- Vanuit 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 exec del_ctry('&landid')
```

Oefening 3

Voorbeeld: 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;
```

Voorbeeld: Andere mogelijke 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;
/
```

Procedure met IN-en OUT parameter(s) oproepen

Een procedure kan worden opgeroepen

- vanuit een andere procedure
- via een anoniem block
- aan de SQL-prompt → bind-variable nodig

Oproeping 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 deze procedure wordt de naam van het department 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 department Administration met een loonsverhoging van 10% worden afgedrukt

Oproeping via een anoniem block

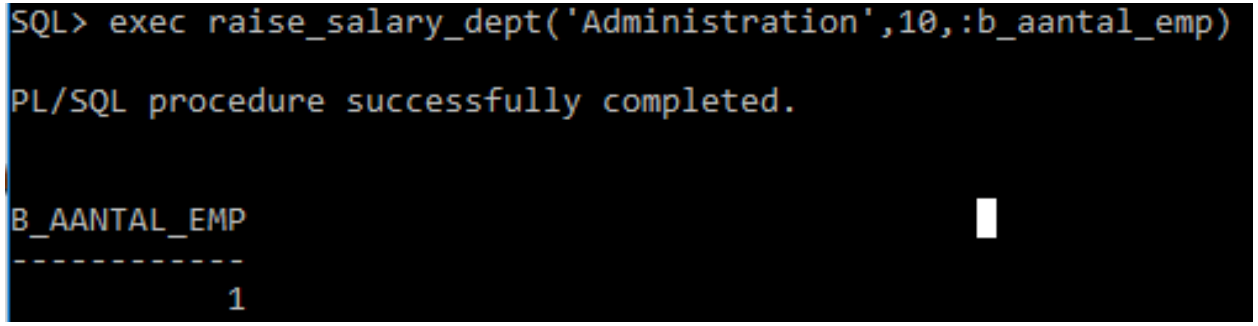
```
DECLARE
        v_aantal_emp        number(3);
BEGIN
        raise_salary_dept('Administration', 10, v_aantal_emp);
        DBMS_OUTPUT.PUT_LINE(v_aantal_emp);
END;
/
```

Oproeping via de SQL-prompt → bind-variable

- Bind variable

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



Oefening

Probeer de procedure raise_salary_2_dept uit te voeren en te gebruiken (Voorbeeld: Andere mogelijke oplossing)

Positional vs. Named notation

- Voorbeeld positional notation
  exec raise_salary_dept(‘Administration’, 10, :b_aantal_emp)

  Volgorde van parameters moet exact dezelfde zijn als in de procedure

- Voorbeeld named notation
  exec raise_salary_dept(p_percent =>10,p_dept_name =>’Administration’,p_count_emp =>:b_aantal_emp)

  Volgorde is niet langer belangrijk maar de parameter-namen moeten dan wel gekend zijn

Oefening 4

Voorbeeld: procedure met eenvoudige LOOP

```
CREATE OR REPLACE PROCEDURE print_dept_loop
AS
        v_count         number(3) := 0;
        v_dept_id       departments.department_id%type;
        v_dept_name     departments.department_name%type;
        v_man_id        departments.manager_id%type;
BEGIN
        LOOP
                v_count := v_count + 10;
                SELECT department_id, department_name, manager_id
                INTO v_dept_id, v_dept_name, v_man_id
                FROM departments
                WHERE department_id = v_count;
                DBMS_OUTPUT.PUT_LINE(v_dept_id||'    '||v_dept_name||'    '||v_man_id);
                EXIT WHEN v_count >=100;
        END LOOP;
END print_dept_loop;
/
```

(herhaal tot …)

Voorbeeld: procedure met WHILE LOOP

```
CREATE OR REPLACE PROCEDURE print_dept_while
AS
        v_count         number(3) := 10;
        v_dept_id       departments.department_id%type;
        v_dept_name     departments.department_name%type;
        v_man_id        departments.manager_id%type;
BEGIN
        WHILE v_count <= 100 LOOP
                SELECT department_id, department_name, manager_id
                INTO v_dept_id, v_dept_name, v_man_id
                FROM departments
                WHERE department_id = v_count;
                DBMS_OUTPUT.PUT_LINE(v_dept_id||'    '||v_dept_name||'    '||v_man_id);
                v_count := v_count + 10;
        END LOOP;
END print_dept_while;
```

(zolang …)

Voorbeeld: procedure met FOR LOOP

```
CREATE OR REPLACE PROCEDURE print_dept_for
AS
        v_dept_id       departments.department_id%type;
        v_dept_name     departments.department_name%type;
        v_man_id        departments.manager_id%type;
BEGIN
        FOR i IN 1..10 LOOP
                SELECT department_id, department_name, manager_id
                INTO v_dept_id, v_dept_name, v_man_id
                FROM departments
                WHERE department_id = i*10;
                DBMS_OUTPUT.PUT_LINE(v_dept_id||'    '||v_dept_name||'    '||v_man_id);
        END LOOP;
END print_dept_for;
/
```

(zelftellende lus)

OPM : teller i wordt impliciet gedeclareerd en kan enkel in lus worden gebruikt – kan wel toegewezen worden aan een variabele en deze is bruikbaar buiten de lus

Oefening 5

Oefening 6

Speciale variant van de FOR-loop – Cursor Loop

```
CREATE OR REPLACE PROCEDURE print_dept_cursorloop
AS
BEGIN
        FOR rec IN (SELECT department_id, department_name, manager_id
                FROM departments
                WHERE department_id between 10 and 100)
                LOOP
                        DBMS_OUTPUT.PUT_LINE(rec.department_id||'    '
                                ||rec.department_name||'   ' ||rec.manager_id);
        END LOOP;
END print_dept_cursorloop;
/
```

Speciale variant van de FOR-loop – Cursor Loop

- Alle rijen en kolommen bekomen door het uitvoeren van de subquery worden in een expliciete cursor bijgehouden.
- Deze cursor heeft geen naam en daarom kan er ook geen gebruik gemaakt worden van cursorattributen
  => rec%rowcount kan NIET GEBRUIKT worden
- De FOR-loop zal rij per rij verwerken
- In de LOOP kan er verwezen worden naar een specifiek attribuut via rec.department_name – het gaat hier dan over de inhoud van het attribuut department_name in de rij die op dat moment door de loop verwerkt wordt

Oefening 7

Oefening 8

Procedures 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 is 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

Opvragen kenmerken (data dictionary)

USER_OBJECTS

belangrijkste kolommen zijn object_name, object_type, created, …

Voorbeeld om te kijken welke procedures aanwezig zijn:

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

Opvragen kenmerken (data dictionary)

USER_SOURCE

belangrijkste kolommen zijn name, type, line, text

Voorbeeld om de code van een bestaande procedure te bekijken:

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

Oefening 9

Oefening 10