Les procédures et les fonctions

En PL/SQL

Béchir RADDAOUI

Année Universitaire 2019/2020

PLAN

DEFINITIONS

1. Une procédure stockée est un **sous-programme** comprenant des instructions **SQL précompilées** et sauvegardé dans le **dictionnaire de données** de la BD.
2. Le plus souvent le programme est écrit dans un langage spécial qui contient à la fois des instructions procédurales (boucles, tests,…) et des ordres SQL.
3. Le langage offert par **Oracle** est le langage **PL/SQL** qui inclut des ordres **SQL** (*certains ayant changé de syntaxe*).
4. Les procédures stockées d'Oracle peuvent aussi être écrites en Java.
5. Une fonction ne renvoie qu'un seul résultat.
6. Une procédure (ou une fonction) peut être exécutée dans un programme principal ou bien sous SQL.
7. Une fonction peut être utilisée comme une variable (*on peut affecter par exemple le résultat d'une fonction dans une variable)*
8. Une procédure peut renvoyer plusieurs résultats
9. Une procédure paramétrée peut avoir plusieurs paramètres de modes différents.
10. Si on ne précise pas le mode d'un paramètre c'est qu'il s'agit d'un paramètre d'entrée (**IN**)

Inconvénient

Le principal inconvénient des procédures et des fonctions est qu'elles impliquent une dépendance forte vis-à-vis du SGBD car chaque SGBD a sa propre syntaxe et son propre langage de programmation.

3. Création d'une procédure stockée

En Oracle, on peut créer une nouvelle procédure stockée par la commande :

CREATE [OR REPLACE] PROCEDURE ***nomprocedure***

[(parameter(s) [mode] type)] **IS**

-- Déclarations des variables locales

BEGIN

-- Instructions

[EXCEPTION]

-- Traitement des exceptions

END nomprocedure;

- **Paramètres**
  - **param** : nom du paramètre
  - **mode** : type de passage du paramètre
    - **IN** (par défaut): entrée (**par valeur**)
    - **OUT** : sortie (**résultat**)
    - **IN OUT** : entrée/sortie (**par référence**)

- **type** : type de donnée du paramètre

4. Exécution d'une procédure

- Une procédure stockée peut être appelée dans un bloc PL/SQL (instructions entre **begin** et **end**) ou bien sous SQL.
- Pour afficher la liste des erreurs, on utilise la commande :

  ❑ **SQL> show errors**

- L'exécution de la commande **CREATE PROCEDURE** stocke le code source dans le dictionnaire de données même si la procédure contient des erreurs de compilation. Il faut donc supprimer la fonction (**DROP**) ou bien utiliser la syntaxe **OR REPLACE** si l'on veut effectuer des changements dans une fonction.
- Pour avoir une description des paramètres d'une procédure stockée

  ❑ **SQL> DESC** *nomprocédure*

- **Exécution à partir de sql*plus**

Une fois la compilation effectuée sans erreur, on utilise **EXECUTE** pour exécuter la procédure.

❑ **SQL> EXECUTE** *nomprocedure(paramètres)*

Exemple 1: paramètre IN

- **Exemple :** d'utilisation du paramètre **IN**: cette procédure augmente le salaire de l'employé ayant pour matricule **v_id** de **10%.**

```
CREATE OR REPLACE PROCEDURE Changer_Salaire
                                (v_id     IN     emp.empno%TYPE)
IS
BEGIN
        UPDATE    emp
        SET       sal = sal * 1.10
        WHERE     empno = v_id;
 COMMIT;
END  Changer_Salaire;
/
```

Procedure created.

Exécution à partir de SQL

```
SQL> EXECUTE Changer_Salaire (7369)
```

PL/SQL procedure successfully completed.

■ Exemple 2 : paramètre OUT

- Ecrire une procédure acceptant un paramètre en entrée (un *numéro d'un employé*) et renvoyant trois résultats. Elle doit renvoyer le ***nom***, le ***salaire*** et la ***commission*** de l'employé ayant pour numéro **v_id**

```
CREATE OR REPLACE PROCEDURE res_emp
(v_id     IN   emp.empno%TYPE, v_name    OUT emp.ename%TYPE,
v_salary  OUT emp.sal%TYPE, v_comm       OUT emp.comm%TYPE)
IS
BEGIN
        SELECT ename, sal, comm   INTO v_name, v_salary, v_comm
        FROM    emp
        WHERE   empno = v_id;
END Res_emp;
/
```

Procedure created.

- **Exécution à partir de SQL**
  - ❑ Il faut déclarer autant de **variables hôtes** que de valeurs retournées par la procédure.
  - ❑ ces variables devront être du même type que les valeurs retournées.

- ensuite ces variables précédées de deux points (:) seront passées en paramètres à la procédure.

■ Exemple d'exécution

Exécution à partir de SQL

SQL> VARIABLE **g_name** VARCHAR2(15)

SQL> VARIABLE **g_sal** NUMBER

SQL> VARIABLE **g_comm** NUMBER

SQL> EXECUTE Res_emp(7654,:g_name,:g_sal,:g_comm)

Procédure PL/SQL terminée avec succès.

SQL> PRINT g_name

G_NAME

BERLAND

■ Exemple 3 : paramètre IN OUT

■ Ecrire une procédure acceptant un paramètre **IN OUT**. Elle transforme un N° de téléphone passé en entrée en une chaîne lisible.

```
CREATE OR REPLACE PROCEDURE format_phone
    (v_phone_no   IN OUT   VARCHAR2(15))
IS
BEGIN
v_phone_no := '(' || SUBSTR(v_phone_no,1,3) || ')' || SUBSTR(v_phone_no,4,2) ||
' ' || SUBSTR(v_phone_no,6,3) ||' '|| SUBSTR (v_phone,9);
END format_phone;
/
```

Procedure created.

■ Exécution à partir de SQL

❑ Pour exécuter une procédure avec un paramètre **IN OUT** il faut préalablement créer et initialiser une variable hôte.

❑ Pour invoquer la procédure FORMAT_PHONE, crée précédemment, dans SQL*Plus on va créer **une variable hôte** par la commande **VARIABLE** puis initialiser celle-ci grâce à un script décrivant un bloc anonyme PL/SQL.

Exécution à partir de SQL

SQL> VARIABLE **g_phone_no** VARCHAR2(15)

SQL> BEGIN :g_phone_no := '21671600444'; **END**;

/ *** *initialisation* ***

SQL> **EXECUTE** FORMAT_PHONE**(:g_phone_no)**

SQL> *Procédure PL/SQL terminée avec succès.*

SQL> PRINT g_phone_no

G_PHONE_NO

---

(216)71 600 444

Dictionnaire des données

Exemple 5 : Les noms des **procédures** stockées et le nom de leur **propriétaire** :

```
SQL>  SELECT    owner, object_name
      FROM      all_objects
      WHERE     object_type = 'PROCEDURE'
                ORDER  BY     owner, object_name;
```

Exemple 6 : Ecrire une procédure ***Afficher_tables*** qui donne le nom de toutes vos tables

```
set serveroutput on
create or replace procedure AFFICHER_tables
 IS

  CURSOR    afficher_nom   IS
                            SELECT   table_name
                            FROM     user_tables;

 BEGIN
    dbms_output.put_line('      Nom des tables : ');

  FOR   nom   IN   afficher_nom
     LOOP
          dbms_output.put_line( nom.table_name);
     END LOOP;
 END;
```

/

Exécution à partir de SQL

- EXECUTE *afficher_tables*;

- Le code de la procédure est donné par :

```
SQL> SELECT   text
       FROM     user_source
      WHERE    name = 'nom-procedure'
      Order  by line;
```

Exemple 7 : Supprimer un département donné

(s'il a moins que trois employés)

```
set serveroutput on;
set verify off;

CREATE OR REPLACE PROCEDURE SUPPRIMER
                    ( N     dept.deptno%type) IS
      nb_emp          number(4);        ndept          number(2);
      inexistant      exception;        plus_deux      exception;
      ville           dept.loc%type;    nomdept        dept.dname%type;

BEGIN
      SELECT  count(*) INTO ndept     FROM dept  WHERE dept.deptno = n;
      SELECT  count(*) INTO nb_emp    FROM emp   WHERE emp.deptno = n;

      IF  ndept=0  then
                        Raise  inexistant;
        else
              IF   nb_emp > 2  then
                        Raise  plus_deux;
            END IF;
      END IF;
      IF  nb_emp = 0  then
          SELECT dname, loc INTO nomdept, ville
            FROM  dept   WHERE  deptno = n;
          INSERT  INTO  DEPTART  VALUES (n, nomdept, ville);
          DELETE  from   dept   WHERE deptno=n;
      END IF;

EXCEPTION
      WHEN  inexistant Then
            DBMS_OUTPUT.PUT_LINE (' le département n''existe pas');
      When  plus_deux Then
            DBMS_OUTPUT.PUT_LINE (' suppression impossible !!!!!');
END;
/
```

Exemple 8: **Afficher le lieu de travail d'un employé de nom donné (*supposé unique*).**

```
set serveroutput on;
Create or replace procedure emp_lieu
        (empname IN     emp.ename%type)
IS
    lieu_travail        dept.dname%type;
BEGIN
    SELECT  loc  INTO   lieu_travail
    FROM    dept
    WHERE  deptno  IN
                (SELECT deptno
                 FROM emp
                 WHERE  ename = empname);
    dbms_output.put_line(empname||'   travaille dans
'|| lieu_travail);
END;
/
```

Exemple 9 : les employés qui travaillent avec un employé donné

```
set serveroutput on

CREATE or replace  PROCEDURE ENSEMBLES ( empnum IN  emp.empno%type )
 IS
        CURSOR   C_ENS    IS
                SELECT     ename
                FROM       emp
                 WHERE   deptno   =   ( SELECT    deptno
                                        FROM     emp
                                        WHERE   empno=empnum );

rec                     C_ENS%rowtype ;
nom_emp                 emp.ename%type;

BEGIN
        SELECT          ename    INTO    nom_emp
        FROM            emp
        WHERE           empno = empnum;

dbms_output.put_line('les employés qui travaillent avec  ' ||nom_emp || ' sont :');

        FOR  rec  IN   C_ENS

        Loop
                IF  rec.ename != nom_emp THEN
                                DBMS_OUTPUT.PUT_LINE(rec.ename);
                END  IF;

        end  loop;
END  ;
/
```

7. Création d'une fonction

Syntaxe :

CREATE [OR REPLACE] FUNCTION ***nomfonction*** (param [mode] type,…) **RETURN Type_retour**

IS

-- Déclarations locales

BEGIN

-- Instructions

[EXCEPTION]

-- Traitement des exceptions

END;

- **Paramètres**
  - **param** : nom du paramètre
  - **mode** : type de passage du paramètre
    - **IN** (***par défaut***): le seul mode autorisé
  - **type** : type de donnée du paramètre
- Une fonction stockée peut être par la suite utilisé dans un ordre SQL à condition qu'elle soit une fonction SINGLE-ROW
- Une fonction stockée peut être appelée dans un bloc PL/SQL (procédure, fonction ou bloc anonyme)
- Suppression d'une procédure stockée

DROP PROCEDURE *nomprocedure* **;**

- Suppression d'une fonction stockée

DROP FUNCTION *nomfonction* **;**

Exemples :

Total des salaires des employés n'appartenant pas au département d'un employé donné

Create or Replace function **TOTAL** (**num_emp** emp.empno%type)
return NUMBER

IS

```
Cursor Emp_Cur Is
  SELECT  Sal
  FROM    emp
  WHERE  deptno < >  (SELECT deptno
                      FROM emp
                      WHERE empno=num_emp);
```

tot_sal emp.sal%type := 0;

BEGIN

FOR emp_record **IN** **emp_cur**

LOOP

tot_sal := tot_sal +emp_record.sal;

END LOOP;

RETURN TOT_SAL;

END total;

/

8. Exécution d'une fonction

---------Programme principal -------

```
set serveroutput on
    ACCEPT num prompt ' Donner un numéro d''employe :  ';
DECLARE
        Somme       emp.sal%type;
        Dnum        dept.deptno%type;
BEGIN
.           Somme := TOTAL(&num);
        SELECT  deptno INTO dnum
        FROM    emp
        WHERE   empno = &num;
DBMS_OUTPUT.PUT_LINE('Les employés n''appartiennent pas au département numéro '||dnum||' gagnent en total :  ' ||somme);
END;
/
```

Exécution à partir de SQL

SQL> VARIABLE **X** number

SQL> **EXECUTE :X** :=total(7369);

Procédure PL/SQL terminée avec succès.

SQL> PRINT X

X

18150

SQL> **EXECUTE** DBMS_OUTPUT.PUT_LINE(**TOTAL(7369**));

18150

Procédure PL/SQL terminée avec succès.

Exemple 11 : Moyenne des salaires d'un département donné

```
set serveroutput on
DECLARE
        v_sal_moy        number(10,3);

FUNCTION  AVG_SAL  (p_deptno  IN  dept.deptno%type)
return  NUMBER
  IS

        moyenne          emp.sal%type;
begin
              SELECT  AVG(sal)  into  moyenne
              FROM    emp
              WHERE deptno = p_deptno;

       RETURN moyenne;

EXCEPTION
        when OTHERS then
                raise_application_error(-20110,'erreur');
end avg_sal;

                *** programme principal ***

BEGIN
        v_sal_moy := AVG_SAL(10);
                dbms_output.put_line('salaire moyen : '||v_sal_moy);

END;
/
```

Exemple 12 : Total des salaires des employés qui sont dans le même département qu'un employé donné

```
Create or replace function revenu_empno(p_empno number) return
number
Is
Cursor Cur_emp is
        SELECT  E2.sal
        FROM    emp E1, emp E2
        WHERE E1.deptno = E2.deptno and
                E1.empno = p_empno;

Revenu          number:=0;

BEGIN

        FOR record_emp IN cur_emp
        loop
                Revenu:=revenu+record_emp.sal;
        End loop;

Return revenu;


END;
/
```

**** Programme principal ****

```
ACCEPT num prompt 'donner le numero employe    ';
DECLARE
        X   number;
BEGIN
       SELECT   empno into X
       FROM     emp
       WHERE   empno = &num;

    dbms_output.put_line('le revenu de ceux qui travaillent avec '||
&num||'   '||revenu_empno(&num));

EXCEPTION
       when no_data_found then
            dbms_output.put_line(' employé inexistant  .....');
END;
/
```

Exécution à partir de SQL

```
SQL> variable   X   number

SQL> EXECUTE   :X:=revenu_empno(7369);
```

Procédure PL/SQL terminée avec succès.

```
SQL> print X

         X
----------
     10875
```

Exemple 13 : Revenu des employés ayant une commission et appartenant au même département d'un employé donné.

```
create or replace function total_revenu(numemp in emp.empno%type)
return NUMBER
IS
        total          number:=0;

CURSOR emp_cursor IS
        SELECT sal, comm
        FROM emp
        WHERE  deptno=(SELECT deptno
                       FROM emp
                       WHERE empno=numemp)
        and comm is not null;

        TYPE emp_record_type IS RECORD
              (com              emp.comm%type,
              Salaire     emp.sal%type
              );
emp_record    emp_record_type;

BEGIN
        OPEN    emp_cursor;
        loop
              FETCH emp_cursor INTO emp_record;
              total:=total+(emp_record.salaire+emp_record.comm);
              EXIT when emp_cursor%notfound;
        end loop;

        CLOSE   emp_cursor;

              RETURN TOTAL;
END;
/
```

EXECUTION

Exécution à partir de SQL

SQL> VARIABLE X number

SQL> EXECUTE X:= total_revenu(**7369**);

Procédure PL/SQL terminée avec succès.

SQL> PRINT X

```
         X
----------
      5950
```

Exercices

Ex1 : Ecrire procédure AJOUT_EMP qui permet d'insérer un employé en faisant les contrôles suivants :

a. le numéro de l'employé doit être unique et non null
b. le numéro de supérieur hiérarchique doit exister dans la table EMP.

Ex2:

Ecrire procédure **P_DEPT**(V_DEPTNO ,V_JOB) qui permet d'afficher le numéro, le nom et le salaire des employés travaillant dans le département V_DEPTNO et ayant la fonction V_Job.

Répondre en utilisant un curseur explicite paramétré (P_Deptno et P_Job) et la boucle de base **LOOP…… END LOOP**.