Procédures et Fonctions PLSQL

Najib Tounsi

Procédures

Une procédure est un bloc nommé, éventuellement paramétré, qu'on peut exécuter à la demande.

Exemple 1 : procédure `bonjour`

```
create or replace procedure bonjour
is
begin
    dbms_output.put_line('Hello');
end;
/
```

On appelle une procédure PLSQL par son nom. On peut le faire directement depuis SQLPlus avec la commande `execute`

```
SQL> execute bonjour;
Hello

PL/SQL procedure successfully completed.
```

ou dans un bloc PLSQL

```
SQL> begin
  2      bonjour;
  3  end;
 /
Hello

PL/SQL procedure successfully completed.
```

Avec paramètre donné `IN`

Exemple 2 : passage de paramètre comme donnée

```
create or replace procedure carre (a in number)
is
begin
     dbms_output.put_line('Carré = '||a*a);
end;
/
Procedure created.
```

Exécution :

```
SQ> execute carre(5)
Carré = 25

PL/SQL procedure successfully completed.
```

Un paramètre `in` est pris comme une donnée et ne doit pas être modifié dans la procédure.

Avec paramètre résultat `OUT`

Exemple 3 : Procédure qui calcule le maximum de deux nombres et le renvoie en résultat

```
declare
   a number;
   procedure max (a in number, b in number, x out number)
   is
     begin
       if a>b then x := a;
       else x:=b;
     end if;
   end;

begin
    max(2,5,a);
    dbms_output.put_line('max = '||a);
end;
/
```

Résultat d'exécution:

```
max = 5

PL/SQL procedure successfully completed
```

Noter sur cet exemple que la procédure `max` a été déclarée à l'intérieur d'un bloc PLSQL dans la zone `declare`.

Paramètres IN OUT

Les paramètres `in out` sont modifiables dans la procédure. Ce sont des paramètres données (en entrée) et résultat (en sortie).

Exemple 4 : Procédure qui permute deux nombres

```
create or replace procedure
  permute (a in out number, b in out number) is
     w number;
   begin
     w := a;
     a := b;
     b := w;
   end;
/
```

Programme test :

```
declare
  x number := 1;
  y number := 2;
begin
  permute (x,y);
  dbms_output.put_line(x ||' '||y);
end;
/
2 1

PL/SQL procedure successfully completed.
```

EXEMPLE COMPLET

Exemple 5 : Augmentation de salaire d'un employé donné avec un taux donné

```
create or replace procedure augmenterSalaire(
    pEnum in employee.enum%type,
    tx in number
) is
begin
   -- Mise à jour salaire employé
   update employee
      set salary = salary + salary * tx / 100
      where enum = pEnum;
end;
```

Programme test : on augmente le salaire des employés gagnant moins qu'un montant.

```
declare
    cursor empCursor is
           select * from EMPLOYEE;
    minSal employee.SALary%TYPE := &smic;
begin
      for employeeRec in empCursor loop
           if  employeeRec.salary < minSal then
                  augmenterSalaire (employeeRec.enum, 10) ;
           end if;
      end loop;
end;
/
```

Exécution :

```
Enter value for smic: 5500
old   4:     minSal employee.SALary%TYPE := &smic;
new   4:     minSal employee.SALary%TYPE := 5500;

PL/SQL procedure successfully completed.

SQL> select * from employee;

ENUM  ENAME                    SALARY ADDRESS    DEPT
----- -------------------- ---------- ---------- -----
E7    Amine                      7500 Fes        D2
E6    Aziz                       8500 Casa       D1
E5    Amina                      8000 Rabat      D3
E4    Said                       5500 Agadir     D3            <--
```

```
E3    Fatima                    7000 Tanger     D2
E2    Ahmed                     6000 Casa       D1
E1    Ali                       8000 Rabat      D1
E8    Ahmed                     4400 Casa       D4            <--
```

Les Fonctions

Les fonctions sont semblables aux procédures, mais retournent une valeur résultat. Une fonction diffère d'une procédure par le fait qu'on peut l'utiliser dans une expression. La fonction suivante convertit un montant en Dirham vers un montant en Euro.

Exemple 6 : convertir un montant en Dirham vers un montant en Euro

```
create function toEuro(montant in number) return number
 is
begin
   return  montant / 10.8;
end;
/

Function created.

SQL> select toEuro(salary) from employee;

TOEURO(SALARY)
--------------
    694.444444
    787.037037
    740.740741
    509.259259
    648.148148
    555.555556
    740.740741
    407.407407

8 rows selected.
```

Noter l'usage dans `select`. Intéressant pour effectuer des calculs sur un résultat d'interrogation.

On rajoute le cas d'erreur de conversion, si le paramètre n'est pas un nombre, auquel cas on retourne `null`.

Exemple 7 : même exemple avec test si le paramètre n'est pas un nombre

```
create or replace function toEuro(montant in varchar2) return number is
begin
   return  to_number (montant) / 10.8;
exception
   when others then            -- si erreur SQL
      return null;
end;
/
```

On teste sur le salaire :

```
SQL> select toEuro(salary) from employee
  2  where enum ='E1';

TOEURO(SALARY)
--------------
    740.740741                    <-- résultat converti
```

On teste sur l'adresse :

```
SQL> select toEuro(address) from employee
  2  where enum ='E1';

TOEURO(ADDRESS)
---------------
                              <-- pas de valeur, résultat null
```

Utilisation dans la clause `where` :

```
SQL> select * from employee where TOEURO(salary)> 600;

ENUM  ENAME                    SALARY ADDRESS    DEPT
----- -------------------- ---------- ---------- -----
E7    Amine                      7500 Fes        D2
E6    Aziz                       8500 Casa       D1
E5    Amina                      8000 Rabat      D3
E3    Fatima                     7000 Tanger     D2
E1    Ali                        8000 Rabat      D1

8 rows selected.
```

On teste si la valeur retour est `null` :

```
SQL> select * from employee where TOEURO(address) is null;

ENUM  ENAME                    SALARY ADDRESS    DEPT
----- -------------------- ---------- ---------- -----
E7    Amine                      7500 Fes        D2
E6    Aziz                       8500 Casa       D1
E5    Amina                      8000 Rabat      D3
E4    Said                       5500 Agadir     D3
E3    Fatima                     7000 Tanger     D2
E2    Ahmed                      6000 Casa       D1
E1    Ali                        8000 Rabat      D1
E8    Ahmed                      4400 Casa       D4

8 rows selected.
```

Gestion des erreurs dans les fonctions et procédures

La procédure Oracle `raise_application_error (codeErreur, message)` permet de soulever des erreurs utilisateurs depuis les procédures ou fonctions enregistrées, et les gérer comme des erreurs Oracles (message `ora-`). La paramètre `codeErreur` (compris entre -20000 et -20999) sera affecté à `SQLCODE` et `message` sera affecté à `SQLERRM`.

On reprend le même programme que l'exemple 5 ci-dessus (augmentation de salaire) et on soulève une erreur si cette augmentation est négative ou indéfinie.

Exemple 8 : soulever une erreur dans une procédure

```
create or replace procedure augmenterSalaire(
    pEnum in employee.enum%type,
    tx in number
```

```
) is
begin
   -- verification
   if tx <0 or tx is null then
       raise_application_error(-20001, 'Augmentation négative ou absente');
   else
     -- Mise à jour salaire employé
     update employee  set salary = salary + salary * tx / 100 where enum = pEnum;
   end if;
end;
/
```

Le programme PLSQL appelant :

```
begin
augmenterSalaire ('E1', &tx);
end;
/
```

donne :

```
SQL> /
Enter value for tx: -6
begin
*
ERROR at line 1:
ORA-20001: Augmentation négative ou absente
 ... autres messages ...

SQL> /
Enter value for tx: null
begin
*
ERROR at line 1:
ORA-20001: Augmentation négative ou absente
... autres messages ....
```

Où on voit que le programme est interrompu et émet l'erreur '`ORA-20001: Augmentation négative ou absente`'

Cela est bien, mais l'intérêt est de récupérer l'erreur et la traiter dans le programme lui même comme exception `others`.

Exemple 8-bis : On ajoute une zone exception au programme précédent pour tester `SQLCODE`.

```
begin
    augmenterSalaire ('E1', -6);
exception
    when others then
      if SQLCODE = -20001 then
          dbms_output.put_line('Une erreur salaire s''est produite : ' || SQLERRM);
      else
           dbms_output.put_line('Autre erreur : ' || SQLERRM);
      end if;
      rollback;
end;
/
```

Si on souhaite définir soi-même une exception pour cette erreur (au lieu de `others`), on doit utiliser la directive de compilation (appelée `PRAGMA` en Oracle) `EXECPTION_INIT`. La ligne

`PRAGMA EXCEPTION_INIT(nomException, codeErreur);`

demande au compilateur d'associer une exception à un code d'erreur.

Exemple 8-ter : On va créer une exception `SalaireNegatif` pour le programme précédent

Nouveau bloc PLSQL :

```
declare
SalaireNegatif exception;
PRAGMA EXCEPTION_INIT(SalaireNegatif, -20001);
begin
  augmenterSalaire ('E1', -6);
exception
    when SalaireNegatif then
        dbms_output.put_line('Une erreur salaire s''est produite : ' || SQLERRM);
        rollback;
end;
/
```

qui produit le même résultat que précédemment.

Dernière màj : Juin 2018