Université Mohammed V
Faculté des Sciences
Rabat

Cours de Bases de données

PL/SQL (Part 3)

Master : Traitement intelligent des systèmes

Préparé par: Mme. Khawla Elansari

Année Universitaire: 2021/ 2022

8. Fonctions & Procedures

Procédures

- Une procédure est un code PL/SQL défini par l'utilisateur et stocké dans la base de données. Ce qui permet d'éliminer la redondance de code.
- Avantages:
  - Le code SQL est précompilé.
  - Exécution plus rapide (puisque stockée dans le serveur)
  - Moins de code redondant
- Syntaxe:

```
CREATE OR REPLACE PROCEDURE schema.NOMProcedure
(param1 [IN|OUT|IN OUT ] typeparam1,
param2 [IN|OUT|IN OUT] Typeparam2, …) AS|IS
   -- Déclarations des variables locales (sans DECLARE)
BEGIN
   -- Bloc PL/SQL
END;
```

Procédures

- CREATE indique que l'on veut créer une procédure.
- OR REPLACE (facultative) permet d'écraser une procédure portant le même nom
- Param1, param2 sont les paramètres de la procédure
- IN :indique que le paramètre transmis par le programme appelant n'est pas modifiable par la procédure. Par défaut les paramètres sont IN
- OUT: indique que les paramètres sont modifiables par la procédure (sortie)
- IN OUT: combinaison de IN et OUT

Procédures

- CREATE indique que l'on veut créer une procédure.
- OR REPLACE (facultative) permet d'écraser une procédure portant le même nom
- Param1, param2 sont les paramètres de la procédure
- IN :indique que le paramètre transmis par le programme appelant n'est pas modifiable par la procédure. Par défaut les paramètres sont IN
- OUT: indique que les paramètres sont modifiables par la procédure (sortie)
- IN OUT: combinaison de IN et OUT

Fonctions

- Les fonctions sont considérées comme des procédures qui retournent des valeurs.
- La seule différence syntaxique par rapport à une procédure se traduit par la présence du mot clé **RETURN**.
- Syntaxe:

```
CREATE OR REPLACE FUNCTION NomFonction
(param1 [IN|OUT|IN OUT ] typeparam1, param2 [IN|OUT|IN OUT] Typeparam2, …)
RETURN TypeVariable AS |IS
  -- Declaration des variables locales (sans DECLARE)
BEGIN
  -- Bloc PL/SQL;
RETURN variable;
END ;
```

Fonctions

```
create or replace function EURtoMAD(montant in number) return number
 is
begin
   return  montant * 10.65;
   dbms_output.put_line(montant * 10.65);
end;

select EURtoMAD(sal) from emp;

select EURtoMAD(98) from dual;
```

9. Packages

Packages

- Un package PL/SQL est un regroupement logique de sous-programmes associés (procédure/fonction) en un seul élément. Un package est compilé et stocké en tant qu'objet de base de données pouvant être utilisé ultérieurement.
- Un package PL/SQL comporte deux composants.
  - Spécifications du Package
  - Corps du Package

Création des spécifications du package

- La spécification de package consiste en une déclaration de toutes les variables publiques, curseurs, objets, procédures, fonctions et exceptions.
- Les éléments qui sont tous déclarés dans la spécification sont accessibles depuis l'extérieur du package. Ces éléments sont appelés éléments publics.
- Syntaxe:

  CREATE [OR REPLACE] PACKAGE <package_name>

  IS

  <sub_program and public element declaration>

  …

  END <package name>

Création des spécifications du package

```
create or replace package gest_emp is
    sal emp.sal%TYPE;
    FUNCTION AugmenterSal (EmpNumber in emp.empno%TYPE, Taux in number) Return Number;
    Procedure Test_AugmenterSal (EmpNumber in emp.empno%TYPE, Taux in number, SalAug out number);
END;
```

Création du corps du package

- Le corps du package consiste en la définition de tous les éléments présents dans la spécification du package. Il peut également contenir une définition d'éléments qui ne sont pas déclarés dans la spécification, ces éléments sont appelés éléments privés et ne peuvent être appelés que depuis l'intérieur du package.
- Syntaxe:

```
CREATE [OR REPLACE] PACKAGE BODY <package_name>
IS
...
END <package_name>
```

Création du corps du package

```
create or replace package body gest_emp is
    empRecord emp%rowtype;
    FUNCTION AugmenterSal (EmpNumber in emp.empno%TYPE, Taux in number) Return Number is
        funcSal emp.sal%TYPE;
        BEGIN
            Select sal into funcsal from emp where empno = EmpNumber;
            funcsal := funcsal * (Taux +1);
            sal := funcsal;
            return funcsal;
        END AugmenterSal;

    PROCEDURE AfficheSal is
        CURSOR empcursor is select * from emp;
        BEGIN
            FOR emprecord IN empcursor LOOP
                dbms_output.put_line(' L''employe ' || emprecord.empno || ' a un salaire de '|| emprecord.sal);
            END LOOP;
        END AfficheSal;
    Procedure Test_AugmenterSal (EmpNumber in emp.empno%TYPE, Taux in number, SalAug out number) is
        procsal emp.sal%TYPE;
        BEGIN
            update emp set sal = sal* (Taux +1);
            AfficheSal;
        END Test_AugmenterSal;
END;
```

Référencer les éléments du package

- Tous les éléments publics du package peuvent être référencés en appelant le nom du package suivi du nom de l'élément séparé par un point.
- Syntaxe:

  <package_name>.<element_name>

```
Declare
    SalAug emp.sal%TYPE;
BEGIN
   SalAug := gest_emp.AugmenterSal(7654, 0.2);
   dbms_output.put_line(salaug);
   dbms_output.put_line(gest_emp.sal); -- Variable globale publique
   gest_emp.Test_AugmenterSal (7654, 0.2, SalAug);
END;
```

10. Déclencheurs

Déclencheurs

- Un **trigger**, également appelé **déclencheur**, permet d'exécuter un ensemble d'instruction SQL juste avant ou après un événement. Cela permet de faciliter et d'automatiser des actions au sein d'un Système de Gestion de Base de Données (SGBD).
- L'événement est en générale une instruction INSERT, UPDATE ou DELETE. Mais cela peut être aussi une commande LDD, un login/logout d'un utilisateur etc. Dans le cas d'une mise à jour, le programme peut se déclencher soit avant que la mise a lieu, soit après.
- Un trigger est donc composé de deux parties : une **entête** et un **corps**. L'entête est l'événement déclencheur et le corps le programme lancé.

Déclencheurs

- Il existe deux types de triggers :
  - **Trigger sur ligne**: le trigger est exécuté pour chaque ligne concernée par l'instruction insert, update ou delete (option "for each row").
  - **Trigger sur instruction**: le trigger est exécuté une seule fois pour l'instruction insert, update ou delete, même si elle traite plusieurs lignes d'un coup.
- Pour les triggers de type "for each row", les colonnes de la ligne courante doivent être référencées spécifiquement selon que l'on veut l' ancienne ou la nouvelle valeur :
  - **:old.**nom_colonne
  - **:new.**nom_colonne

Déclencheurs

- CREATE TRIGGER nom du trigger

  {BEFORE | AFTER } événement déclencheur ON table mise à jour

  [... options supplémentaires...]

  [DECLARE

  déclarations]

  BEGIN

  instructions

  [EXCEPTION

  prise en compte des exceptions]

  END;

Thank you!