Compléments de bases de données

COURS 2 :

PL/SQL

Novembre 28, 2019

ROMDHANI Senda
senda.romdhani@insa-lyon.fr

*** Cours extraits principalement des cours de Guillaume Cabanac, Fouad Dahak, Yacine Bouhabel, Amrouche Karima, Richard Grin**

Pourquoi PL/SQL= PROCEDURAL LANGUAGE/SQL ?

- SQL (Structured Query Language) est un langage permettant de communiquer avec une base de données.
  - ✓Configurer la base de données.
  - ✓La remplir avec des données.
  - ✓Effectuer des “requêtes” simples pour récupérer les données
- SQL est un langage non procédural
- S’il vous faut rédiger une logique conditionnelle très complexe ou des boucles spécialisées sur les enregistrements d’un ensembles de données, il est possible que SQL ne suffise pas.

➡ On ressent vite le besoin d’un langage procédural pour lier plusieurs requêtes SQL avec des structures de programmation habituelles.

SOMMAIRe

SOMMAIRe

1. Introduction

Principales caractéristiques

- PL/SQL est un langage qui intègre SQL et permet de programmer d'une manière procédurale. (un langage procédurale contient simplement une série d'étapes à réaliser)
- PL/SQL est un langage spécifique à Oracle.
- Un programme constitué de procédures et de fonctions.
- Des variables permettant l'échange d'informations entre les requêtes SQL et le reste du programme.

1. Introduction

Utilisation de PL/SQL

- PL/SQL peut être utilisé pour l'écriture des procédures stockées et des triggers (déclencheurs: déclencher l'exécution d'une instruction).
- Il convient aussi pour écrire des fonctions utilisateurs qui peuvent être utilisées dans les requêtes SQL (en plus des fonctions prédéfinies).
- Il est aussi utilisé dans des outils Oracle
    - Ex: Forms et Report

SOMMAIRe

- Introduction
- Éléments de base
- Curseurs
- Exceptions
- Procédures et fonctions
- Déclencheurs (Triggers)

2. Eléments de base

Blocs PL/SQL

- Un programme est structuré en blocs d'instructions de 3 types :

Anonyme

```
DECLARE
    -variables
BEGIN
    -code du programme
EXCEPTION
END;
```

Procédure nommée

```
PROCEDURE <name>
IS
BEGIN
    -code du programme
EXCEPTION
END;
```

Fonction nommée

```
FUNCTION <name>
RETURN <datatype>
BEGIN
    -code du programme
EXCEPTION
END;
```

NB: Un bloc peut contenir d'autres blocs et il doit contenir en moins une instruction

2. Eléments de base

Structure d'un bloc

```
DECLARE
      -- définition des variables
BEGIN
      -- code du programme
EXCEPTION
      -- code de gestion des erreurs
END;
```

- Seuls BEGIN et END sont obligatoires.
- Les blocs se terminent par un ;

2.1. Déclaration

Structure d'une déclaration

```
nom-variable type-variable [ := valeur ] ;
```

Exemples

```
age number(10);
nom varchar(30) := 'Dupont';
date_naissance date;
ok boolean := true;
v_x,v_y VARCHAR2(30);
```

Variables implicitement initialisées à NULL.

ne fonctionne par car il faut un seul identifiant par ligne.

2.1. Déclaration

L'identificateur nom-variable respecte les contraintes suivantes:

- Il doit être déclaré pour pouvoir l'utiliser.
- 30 caractères maximum
- Il doit commencer par une lettre
- Il peut contenir lettres, chiffres, _, $, et #
- Il n'est pas sensible à la casse (n'est pas sensible maj/min).

2.1. Déclaration

Structure d'une déclaration

```
nom-variable type-variable [ := valeur ] ;
```

`type-variable` peut être :

- **Type simple SQL** VARCHAR2, NUMBER, DATE, BOOLEAN
- **Type d'une colonne d'une table** %TYPE

```
Exemple : v_nom employe.nom%TYPE
```

- **Type d'une ligne entière d'une table** %ROWTYPE

```
Exemple : v_employe employe%ROWTYPE
```

2.2.1 Commandes principales - affectations

Structure d'une déclaration

```
nom-variable type-variable [ := valeur ] ;
```

Affectation d'une valeur à une variable

- Opérateur d'affectation « := » .

  Exemple : `v_nom := 'Rogers';`

- Directive INTO de la requête SELECT.

  ```
  Exemple: SELECT nom INTO v_nom FROM employe where dept=50;
  ```

- Cette commande récupère le résultat du SELECT et l'affecte à la variable v_nom

2.2.1 Commandes principales - affectations

Exemple d'utilisation

```
DECLARE
      v_emp emp%ROWTYPE;
BEGIN
      SELECT * INTO v_emp FROM emp WHERE dept=50;
     v_emp.dept=20
      INSERT INTO emp VALUES v_emp;
END;
```

Pour éviter les conflits de nommage, préfixer les variables PL/SQL par v_

2.2.2 Commandes principales - conditions

Syntaxe IF-THEN

```
IF condition THEN
   instructions…
END IF;
```

retourner un résultat disponible avec une seule possibilité.

Syntaxe IF-THEN-ELSE

```
IF condition THEN
   instructions…
ELSE
   instructions…
END IF;
```

retourner un résultat disponible entre deux possibilités.

Syntaxe IF-THEN-ELSIF

```
IF condition_1 THEN
   instructions…
ELSIF condition_2 THEN
   instructions…
END IF;
```

retourner un résultat disponible entre plusieurs possibilités.

Syntaxe CASE

```
CASE selecteur
   WHEN expression1 THEN resultat1
   WHEN expression2 THEN resultat2
   ELSE resultat3
END;
```

2.2.2 Commandes principales - conditions

Exemple IF-THEN

```
IF v_date>'11-APR-03' THEN
   v_salaire:= v_salaire*1.15;
END IF;
```

Exemple IF-THEN-ELSIF

```
IF v_nom='Paker' THEN
 v_salaire:= v_salaire*1.15;
ELSIFv_nom='ASTROFF'
 v_salaire:= v_salaire*1.05;
END IF;
```

Exemple IF-THEN-ELSE

```
IF v_date>'11-APR-03' THEN
   v_salaire:= v_salaire*1.15;
ELSE
   v_salaire:= v_salaire*1.05;
END IF;
```

Exemple CASE

```
val :=CASE city
   WHEN 'TORONTO' THEN 'RAPTORS'
   WHEN 'LOS ANGELES' THEN 'LAKERS'
   ELSE 'NO TEAM'
END;
```

2.2.2 Commandes principales - boucles

Syntaxe LOOP

```
LOOP
    instructions…
    EXIT [WHEN condition];
END LOOP;
```

Obligation d'utiliser la commande EXIT pour éviter une boucle infinie.

Syntaxe WHILE

```
WHILE condition LOOP
   instructions…
END LOOP;
```

Tant que condition est vraie, les instructions sont répétées

Syntaxe FOR

```
FOR variable IN debut..fin  LOOP
    instructions…
END LOOP;
```

variable prend les valeurs de debut, debut+1,…, jusqu'à fin. Ne pas déclarer variable, elle est déclarée implicitement.

2.2.2 Commandes principales - boucles

Exemple LOOP

```
v_cpt:= 1;
LOOP
   update emp set salaire:=salaire*2
   where emp_id=v_cpt;
   v_cpt:= v_cpt+1;
   EXIT WHEN v_cpt>10;
END LOOP
```

Exemple WHILE

```
v_cpt:= 1;
WHILE v_cpt<=10 LOOP
   update emp set salaire:=salaire*2
   where emp_id=v_cpt;
   v_cpt:= v_cpt+1;
END LOOP
```

Exemple FOR

```
FOR v_cpt in 1..10 LOOP
   update emp set salaire:=salaire*2
   where emp_id=v_cpt;
END LOOP
```

2.2.2 Commandes principales - affichage

Syntaxe d'affichage

```
DBMS_OUTPUT.PUT_LINE(chaine);
```

Remarques

- Avant d'utiliser cette instruction, il faut initialiser le buffer de sortie : SET SERVEROUTPUT ON SIZE buffersize.
- Utiliser || pour faire des concaténations.

Exemple d'utilisation

```
SERVEROUTPUT ON SIZE buffersize;
DECLARE
   v_nom varchar2(25);
BEGIN
   SELECT nom INTO v_nom FROM emp
   WHERE id=50;
   DBMS_OUTPUT.PUT_LINE('le nom est
   : ' || v_nom);
END;
```

2.2.2 Commandes principales - commentaires

Syntaxe d'un commentaire

```
-- Pour une fin de ligne

/* Pour plusieurs lignes */
```