Fondements des bases de données

Programmation en PL/SQL Oracle (1/2)

Équipe pédagogique BD

romuald.thion@univ-lyon1.fr

http://liris.cnrs.fr/~rthion/dokuwiki/enseignement:lif10/

Version du 12 février 2014

Langage PL/SQL

Commandes

Curseurs

Pourquoi PL/SQL ?

PL/SQL = PROCEDURAL LANGUAGE/SQL

- SQL est un langage non procédural
- Les traitements complexes sont parfois difficiles à écrire si on ne peut utiliser des variables et les structures de programmation comme les boucles et les alternatives
- On ressent vite le besoin d'un langage procédural pour lier plusieurs requêtes SQL avec des variables et dans les structures de programmation habituelles

Principales caractéristiques

- Extension de SQL : des requêtes SQL cohabitent avec les structures de contrôle habituelles de la programmation structurée (blocs, alternatives, boucles).
- La syntaxe ressemble au langage Ada ou Pascal.
- Un programme est constitué de procédures et de fonctions.
- Des variables permettent l'échange d'information entre les requêtes SQL et le reste du programme

Utilisation de PL/SQL

- PL/SQL peut être utilisé pour l'écriture des procédures stockées et des triggers.
  - (Oracle accepte aussi le langage Java)
- 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*.

Utilisation de PL/SQL (suite)

Le PL/SQL peut être utilisé sous 3 formes :

1. un bloc de code, exécuté comme une unique commande SQL, via un interpréteur standard (SQLplus ou iSQL*PLus)
2. un fichier de commande PL/SQL
3. un programme stocké (procédure, fonction, trigger)

Blocs

- Un programme est structuré en blocs d'instructions de 3 types :
  - procédures ou bloc anonymes,
  - procédures nommées,
  - fonctions nommées.
- Un bloc peut contenir d'autres blocs.
- Considérons d'abord les blocs anonymes.

Structure d'un bloc anonyme

```
DECLARE
   -- definition 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 ;

Déclaration, initialisation des variables

- Identificateurs Oracle :
  - 30 caractères au plus,
  - commence par une lettre,
  - peut contenir lettres, chiffres, _, $ et #
  - pas sensible à la casse.
- Portée habituelle des langages à blocs
- Doivent être déclarées avant d'être utilisées

- Déclaration et initialisation
  - `Nom_variable type_variable := valeur;`
- Initialisation
  - `Nom_variable := valeur;`
- Déclarations multiples interdites.

Exemples

- `age integer;`
- `nom varchar(30);`
- `dateNaissance date;`
- `ok boolean := true;`

Plusieurs façons d'affecter une valeur à une variable

- Opérateur d'affectation `n:=`.
- Directive `INTO` de la requête `SELECT`.

Exemples

- ```
  dateNaissance :=
  to_date('10/10/2004','DD/MM/YYYY');
  ```
- ```
  SELECT nom INTO v_nom
  FROM emp
  WHERE matr = 509;
  ```

Attention

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

SELECT ...INTO ...

Instruction **SELECT** `expr1,expr2, ...` **INTO** `var1, var2, ...`

- Met des valeurs de la BD dans une ou plusieurs variables `var1, var2, ...`
- Le `SELECT` ne doit retourner qu'une seule ligne
- Avec Oracle il n'est pas possible d'inclure un `SELECT` sans `INTO` dans une procédure.
- Pour retourner plusieurs lignes, voir la suite du cours sur les curseurs.

Types de variables

VARCHAR2

- Longueur maximale : 32767 octets ;
- Exemples :

```
name VARCHAR2(30);
name VARCHAR2(30) := 'toto';
```

NUMBER(long,dec)

- Long : longueur maximale ;
- Dec : longueur de la partie décimale ;
- Exemples :

```
num_telnumber(10);
toto number(5,2)=142.12;
```

DATE

- Fonction TO_DATE ;
- Exemples :

```
start_date :=
to_date('29-SEP-2003','DD-MON-YYYY');
start_date :=
to_date('29-SEP-2003:13:01','DD-MON-YYYY:HH24:MI');
```

BOOLEAN

- TRUE
- FALSE
- NULL

Déclaration %TYPE et %ROWTYPE

```
v_nom emp.nom.%TYPE;
```

☛ On peut déclarer qu'une variable est du même type qu'une colonne d'une table ou (ou qu'une autre variable).

```
v_employe emp%ROWTYPE;
```

☛ Une variable peut contenir toutes les colonnes d'un tuple d'une table (la variable `v_employe` contiendra une ligne de la table `emp`).

Important pour la robustesse du code

Exemple

```
DECLARE
   -- Declaration
   v_employe emp%ROWTYPE;
   v_nom emp.nom.%TYPE;
BEGIN
   SELECT * INTO v_employe
   FROM emp
   WHERE matr = 900;
   v_nom := v_employe.nom;
   v_employe.dept := 20;
...
   -- Insertion d'un tuple dans la base
   INSERT into emp VALUES v_employe;
END;
```

Vérifier à bien retourner *un seul tuple*
avec la requête `SELECT ...INTO ...`

Test conditionnel

IF-THEN

```
IF v_date > '01-JAN-08' THEN
  v_salaire := v_salaire * 1.15;
END IF;
```

IF-THEN-ELSE

```
IF v_date > '01-JAN-08' THEN
  v_salaire := v_salaire * 1.15;
ELSE
  v_salaire := v_salaire * 1.05;
END IF;
```

IF-THEN-ELSIF

```
IF v_nom = 'PARKER' THEN
  v_salaire := v_salaire * 1.15;
ELSIF v_nom = 'SMITH' THEN
  v_salaire := v_salaire * 1.05;
END IF;
```

CASE

```
CASE selection
  WHEN expression1 THEN resultat1
  WHEN expression2 THEN resultat2
   ...
  ELSE resultat
END;
```

CASE renvoie une valeur qui vaut resultat1 ou resultat2 ou ...ou resultat par défaut.

Exemple

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

Les boucles

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

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

Exemple

```
LOOP
    monthly_value := daily_value * 31;
    EXIT WHEN monthly_value > 4000;
END LOOP;
```

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

FOR

```
FOR variable IN [REVERSE] debut..fin
LOOP
    instructions;
END LOOP;
```

- La variable de boucle prend successivement les valeurs de *debut*, *debut* + 1, *debut* + 2, . . ., jusqu'à la valeur *fin*.
- On pourra également utiliser un curseur dans la clause `IN` (dans quelques slides).
- Le mot clef `REVERSE` à l'effet escompté.

Exemple

```
FOR Lcntr IN REVERSE 1..15
LOOP
    LCalc := Lcntr * 31;
END LOOP;
```

Affichage

- Activer le retour écran : `set serveroutput on size 10000`
- Sortie standard : `dbms_output.put_line(chaîne);`
- Concaténation de chaînes : opérateur `||`

Exemple

```
DECLARE
   i number(2);
BEGIN
   FOR i IN 1..5 LOOP
      dbms_output.put_line('Nombre: ' || i);
   END LOOP;
END;
/
```

Le caractère / seul sur une ligne déclenche l'évaluation.

Affichage

Exemple bis

```
DECLARE
   compteur number(3);
   i number(3);
BEGIN
   SELECT COUNT(*) INTO compteur
   FROM EtudiantLIF10;

   FOR i IN 1..compteur LOOP
      dbms_output.put_line('Nombre : L3IF ' || i );
   END LOOP;
END;
```

Langage PL/SQL

Commandes

Curseurs

Les curseurs

Toutes les requêtes SQL sont associées à un curseur :

- Ce curseur représente la zone mémoire utilisée pour analyser et exécuter la requête.
- Le curseur peut être implicite (pas déclaré par l'utilisateur) ou explicite.
- Les curseurs explicites permettent de manipuler l'ensemble des résultats d'une requête.

Les curseurs *implicites* sont tous nommés `SQL`

```
DECLARE
  nb_lignes integer;
BEGIN
  DELETE FROM emp WHERE dept = 10;
  nb_lignes := SQL%ROWCOUNT;
  ...
END;
```

Attributs des curseurs

Tous les curseurs ont des attributs que l'utilisateur peut utiliser :

`%ROWCOUNT` Nombre de lignes traitées par le curseur.

`%FOUND` Vrai si au moins une ligne a été traitée par la requête ou le dernier fetch.

`%NOTFOUND` Vrai si aucune ligne n'a été traitée par la requête ou le dernier fetch.

`%ISOPEN` Vrai si le curseur est ouvert (utile seulement pour les curseurs explicites)

Les curseurs explicites

Pour traiter les SELECT qui renvoient plusieurs lignes

- Les curseurs doivent être déclarés explicitement. A la déclaration, on explicite la requête SELECT dont le résultat sera parcouru par le curseur.
- Le code doit les utiliser avec les commandes
  - `OPEN moncurseur`, pour ouvrir le curseur ;
  - `FETCH moncurseur`, pour avancer le curseur à la ligne suivante ;
  - `CLOSE moncurseur`, pour refermer le curseur

Utilisation

- On utilise souvent les curseurs dans une boucle FOR qui permet une utilisation *implicite* des instructions OPEN, FETCH et CLOSE.
- Généralement, on sort de la boucle quand l'attribut NOTFOUND est vrai.

Les curseurs explicites

Exemple de boucle LOOP pour les curseurs

```
DECLARE
CURSOR c_salaires IS
      SELECT sal
      FROM emp;
v_salaire number(4);
v_total number(6);
BEGIN
   v_total := 0;
   OPEN c_salaires;
   LOOP
     FETCH c_salaires INTO v_salaire;
   EXIT WHEN c_salaires%NOTFOUND;
     IF v_salaire IS NOT NULL THEN
        v_salaire := v_total + v_salaire;
        dbms_output.put_line(total);
     END IF;
   END LOOP;
   CLOSE c_salaires;
   dbms_output.put_line(v_total);
END;
```

Les curseurs explicites

Déclaration d'un type associé à un curseur

```
DECLARE
   CURSOR c IS
      SELECT matr, nom, sal
      FROM emp;
   employe c%ROWTYPE;
BEGIN
   OPEN c;
   FETCH c INTO employe;
   IF employe.sal IS NOT NULL THEN
   ...
   END IF;
END;
```

Boucle FOR pour un curseur

- Elle *simplifie* la programmation car elle évite d'utiliser explicitement les instruction OPEN, FETCH et CLOSE.
- En plus elle déclare *implicitement* une variable de type ROW associée au curseur.

Exemple

```
DECLARE
  CURSOR c_nom_clients IS
  SELECT nom, adresse
  FROM clients;
BEGIN
  FOR le_client IN c_nom_clients LOOP
    dbms_output.put_line(
      'Employe : ' || UPPER(le_client.nom) ||
      ' Ville : ' || le_client.adresse);
  END LOOP;
END;
```

Préfixer le nom d'un curseur par c_ pour éviter les confusions

Curseurs paramétrés

- Un curseur paramétré peut servir plusieurs fois avec des valeurs des paramètres différentes.
- On doit fermer le curseur entre chaque utilisation de paramètres différents si on utilise pas la boucle `FOR` dédiée.

Exemple

```
DECLARE
  CURSOR c(p_dept integer) IS
    SELECT dept, nom
    FROM emp
    WHERE dept = p_dept;
BEGIN
  FOR employe in c(10) LOOP
  dbms_output.put_line(employe.nom);
END LOOP;
FOR employe in c(20) LOOP
  dbms_output.put_line(employe.nom);
END LOOP;
END;
```

Fin du cinquième cours.