Fondements des bases de données

Programmation en PL/SQL Oracle – commandes de base

Équipe pédagogique BD

`http://liris.cnrs.fr/~mplantev/doku/doku.php?id=lif10_2016a`

Version du 13 octobre 2016

Langage PL/SQL

Commandes

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*.

Documentations de références

- http://dba.stackexchange.com/
- http://docs.oracle.com/cd/E11882_01/
- http://www.techonthenet.com/oracle/
- http://www.w3schools.com/sql/
- http://en.wikibooks.org/wiki/Oracle_Programming/SQL_Cheatsheet

Langage PL/SQL

Commandes

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 = 0;
   v_nom := v_employe.nom;
   v_employe.dep := 20;
   v_employe.matr := 99;
   -- ...
   -- 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 ...`

Langage PL/SQL

Commandes

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 EMP;

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

Fin du cours.