PL/SQL

Plan PL/SQL

- Introduction
- Bloc PL/SQL
- Déclaration des variables
- Structure de contrôle
- Curseurs
- Les exceptions
- Les fonctions et procédures
- Les packages
- Les triggers

Introduction

Nécessité de PL/SQL

- **PL/SQL**: Procedural language/SQL
- **SQL** est un langage non procédural
- Parfois les traitements complexes sont difficiles à écrire, si l'on ne peut pas utiliser les variables et les structures de programmation comme les boucles, les conditions, procédures, ;….
- Pour cette raison, on doit avoir un langage procédural pour combiner des requêtes SQL avec des variables et des structures de programmation habituelles.

Introduction (2)

Quelques caractéristiques de PL/SQL

- Extension du langage SQL : des requêtes SQL se combinent avec les structures de contrôle habituelles de la programmation structurée (blocs, boucles,…)
- Sa syntaxe ressemble à celle du langage Pascal et Ada
- Un programme est constitué de procédures, de fonctions,…
- En général, l'échange d'information entre les requêtes SQL et le reste du programme est effectué par des variables.
- PL/SQL est un langage propriétaire de Oracle. Il est créé par Oracle et utilisé dans le cadre de bases de données relationnelles.

Introduction (3)

Quelques caractéristiques de PL/SQL (suite)

PL/SQL : langage procédural d'Oracle étend SQL

- ✓ Instructions SQL intégrées dans PL/SQL
- ✓ Instructions spécifiques à PL/SQL

Instructions SQL intégrées dans PL/SQL

- ➢ Instructions du **LRD** : SELECT
- ➢ Instructions de **LDD** : CREATE, ALTER, DROP, RENAME, TRUNCATE
- ➢ Instructions du **LMD** : INSERT, UPDATE, DELETE
- ➢ Instructions du langage de contrôle des transactions (**LCT**) : COMMIT, ROLLBACK, SAVEPOINT,…
- ➢ Fonctions: TO_CHAR, TO_DATE, UPPER, SUBSTR, …

Introduction (4)

Instructions spécifiques à PL/SQL

- ✓ Définition de variables
- ✓ Traitements conditionnels
- ✓ Traitement répétitifs (ou boucles)
- ✓ Traitement des curseurs
- ✓ Traitement des erreurs
- ✓ …

Introduction (5)

Base et utilisation de PL/SQL

- PL/SQL : langage basé sur les paradigmes de programmation procédurale et structurée.
- IL est utilisé pour l'écriture des procédures stockées et des déclencheurs (triggers).
- On l'utilise aussi pour écrire des fonctions utilisateurs qui peuvent être exploitées dans les requêtes SQL, ainsi que pour des fonctions prédéfinies.
- On l'utilise aussi dans plusieurs outils d'Oracle: *Forms, Report,…*

Structure d'un programme

Unité de base : blocs

- PL/SQL n'interprète pas une commande, mais un ensemble de commandes contenues dans un bloc PL/SQL.
- En général, un programme est organisé en blocs d'instructions de 3 types :
  - Procédures anonymes
  - Procédures nommées
  - Fonctions nommées
- Un bloc peut contenir plusieurs autres blocs.

Structure d'un programme (2)

Structure d'un bloc

DECLARE -- Section optionnelle

-- définitions de variables

BEGIN -- Section obligatoire

--implémentation : Les instructions à exécuter

EXCEPTION -- Section optionnelle

-- code de gestion des erreurs

END;

Seuls BEGIN et END sont obligatoires

Les blocs, comme les instructions, se terminent par un « ; »

N.B: Le programme peut être:

- Directement tapé sur une ligne de commande ou
- Ecrit dans un fichier puis chargé

Variables

Variables en pl/SQL

- Identificateur Oracle :
  - Comporte 30 caractères au plus,
  - Commence par une lettre,
  - Peut contenir des lettres, des chiffres, _, $ et #
- Pas sensible à la casse
- Portée habituelle des langages à blocs
- Doit être déclarée avant d'être utilisée

Variables (2)

Possibilités de placer les commentaires

- **--** Commentaire sur une ligne
- /* commentaire sur plusieurs lignes */

Types de variables

- Types habituels qui correspondent aux types SQL2 ou Oracle : **integer, number, varchar,…**
- Types composites qui s'adaptent à la récupération des lignes, des colonnes et des tables SQL : **%TYPE, %ROWTYPE**
- Type référence : **REF**

Variables (3)

Déclaration et initialisation d'une Variable

- identificateur (**CONSTANT**] type **:=** valeur;
- **Exemples** :
  - age **integer**;
  - nom **varchar(30)**;
  - dateNaissance **date**;
  - ok **boolean** := true;
- Déclarations multiples **interdites** :

i, j integer; //**interdit**

Variables (4)

Déclaration %TYPE

- Possible de déclarer qu'une variable est du même type qu'une colonne d'une table ou autre variable (ou qu'une vue) :

nom emp.nome%TYPE;

Déclaration %ROWTYPE

- Une variable peut contenir toutes les colonnes d'une ligne d'une table.

employe emp%ROWTYPE;

Variables (5)

Exemple

Considérant la table: **emp** (empno, ename, fonction, mgr, date_embauche, sal, comm, deptno)

employe emp**%ROWTYPE**;

nom emp.ename**%TYPE**;

select * **INTO** employe **from** emp **where** ename= ‘Alaoui’;

nom := employe.ename;

employe.deptno := 20;// emp a un attribut deptno

…

insert into emp **values** employe;

Variables (6)

Type RECORD

- Equivalent à **struct** du langage C
- **TYPE** *nomRecord* ***IS*** **RECORD** *(*

  champ1 type1,

  champ2 type2,

  Champ3 type 3;

  …

);

Variables (7)

Exemple

```
Declare
   TYPE enreg IS RECORD (
    num emp.empno%TYPE,
    nom emp.ename%TYPE,
    job emp.job%TYPE );
    R_EMP enreg ; -- variable record  de type enreg
Begin
    R_EMP.num := 1 ;
    R_EMP.nom := ‘Idrissi' ;
    R_EMP.job := ‘Ingénieur' ;
End;
```

emp est une table contient les champs: empno, ename,…

Variables (8)

Affectation

- Plusieurs façons de donner une valeur à une variable :
  - Opérateur d'affectation: (**:=**)
  - Directive **INTO** de la requête **SELECT**

Exemples :

- Date_embauche := '10/10/2004';
- **select** ename **INTO** nom **from** emp **where** empno= 3;
  - **Select** ne renvoie qu'une seule ligne,
  - Dans Oracle, il n'est pas possible d'inclure la clause **Select** sans **into** dans une procédure.
  - Pour renvoyer plusieurs lignes, voir la suite du cours dans les curseurs

Variables (9)

Problèmes de conflits de noms

- Si une variable porte le même nom qu'une colonne d'une table, c'est la colonne qui l'emporte.

```
DECLARE
   ename varchar(30) := 'Samri';
BEGIN
delete from emp where ename='Abdi';//emp contient un champ ename
END;
```

- Pour éviter les conflits de nommages, préfixer les variables PL/SQL par **v_**

Variables (10)

Affichage

- Pour plus de clarté, il est utile d'afficher les valeurs des variables;
- Activer le retour écran: **set serveroutput on**
- Sortie standard; le paquetage: **DBMS_OUTPUT**
- Un paquetage est un regroupement de procédures et de fonctions, voir la suite du cours.
- Concaténation de chaîne: opérateur **||**

Variables (11)

Exemple 1

```
set serveroutput on
        a number;
begin
        a:=10;
        dbms_output.put_line('La valeur de a est: ' || a);
end;
```

Résultat: La valeur de a est : 10

Variables (12)

Exemple 2

```
set serveroutput on
        nb number;
begin
        delete from emp where ename='Snoussi';
        nb := sql%rowcount; -- curseur sql
        dbms_output.put_line('nb = ' || nb);
end;
```

Résultat: retourne le nombre d'enregistrement supprimés dont le champ nom est '**Snoussi'**

Structures de contrôle

Alternative

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

```
IF condition THEN
        instructions1;
ELSE
        instructions2;
END IF;
```

```
IF condition1 THEN
        instructions 1;
ELSIF condition2 THEN
        instructions2;
ELSIF …
        …;
ELSE
        instructionsN;
END IF;
```

Structures de contrôle (2)

Exemple

ACCEPT n **number PROMPT** 'Veuillez saisir un nombre: '

```
set serveroutput on
declare
        n number;
begin
        n:=3;
        If (n>0) then dbms_output.put_line('n est strictement positif');
        elsif (n=0) then dbms_output.put_line('n est null');
        Else dbms_output.put_line('n est strictement négatif');
        end if;
end;
```

n:=&n

Structures de contrôle (3)

Choix

- **CASE** expression

  WHEN expr1 **THEN** instructions1;

  WHEN expr2 **THEN** instructions2;

  …

  ELSE instructionsN;

  END CASE;

- Expression peut avoir n'importe quel type simple (ne peut pas par exemple être un **RECORD**)

Structures de contrôle (4)

Exemple

```
set serveroutput on
declare
        n integer;
begin
         n:=3;
CASE n
        WHEN 1THEN dbms_output.put_line('Lundi');
        WHEN 2THEN dbms_output.put_line('Mardi');
        WHEN 3THEN dbms_output.put_line('Mercredi');
        WHEN 4THEN dbms_output.put_line('Jeudi');
        WHEN 5THEN dbms_output.put_line('Vendredi');
        WHEN 6THEN dbms_output.put_line('Samedi');
        ELSE dbms_output.put_line('Dimanche');
 END CASE;
END;
```

Structures de contrôle (5)

Boucle « tant que »

WHILE condition **LOOP**

instructions;

END LOOP;

Structures de contrôle (6)

Exemple: calcul de la moyenne de 10 entiers

```
SET SERVEROUTPUT ON
DECLARE
        compteur NUMBER(2);
        somme NUMBER(2) := 0;
        moyenne NUMBER(3,1);
BEGIN
    compteur := 1;
    WHILE compteur <= 10 LOOP
        somme := somme + compteur ;
        compteur := compteur + 1;
    END LOOP;
    moyenne := somme / 10;
    DBMS_OUTPUT.PUT_LINE ('La moyenne est '||moyenne);
END;
```

Structures de contrôle (7)

Boucle «Faire … tant que »

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

Structures de contrôle (8)

Exemple: calcul de la moyenne de 10 entiers

```
 SET SERVEROUTPUT ON
DECLARE
        compteur NUMBER(2);
        somme NUMBER(2) := 0;
        moyenne NUMBER(3,1);
BEGIN
    compteur := 1;
    LOOP
        somme := somme + compteur ;
        compteur := compteur + 1;
    Exit when compteur > 10;
    END LOOP;
    moyenne := somme / 10;
    DBMS_OUTPUT.PUT_LINE ('La moyenne est '|| moyenne);
END;
```

Structures de contrôle (9)

Boucle « pour »

```
FOR compteur IN inf..sup LOOP
        instructions;
END LOOP;
```

Exemple :

```
FOR i IN 1..100 LOOP
        somme := somme + i;
END LOOP;
```

Interactions simples avec la base

Extraire des données – erreurs

- Si le **select** renvoie plus d'une ligne, une exception « **TOO_MANY_ROWS » (ORA-01422)** est levée. Voir la suite du cours sur les exceptions.
- Si le select ne renvoie aucune ligne, une exception « **NO_DATA_FOUND » (ORA-01403)** est levée.

Interactions simples avec la base (2)

Exemple

```
DECLARE
        v_nom emp.nome%TYPE;// nome: un champ de emp
        v_emp emp%ROWTYPE;
BEGIN
        select nome into v_nom from emp where matr = 500;
        select * into v_emp from emp where matr = 500;
END;
```

Interactions simples avec la base (3)

Modification de données

- Les requêtes SQL (**insert**, **update**, **delete**,…) peuvent utiliser les variables PL/SQL
- Les **commit** et **rollback** doivent être explicites ; aucun n'est effectué automatiquement à la sortie d'un bloc.
- Voyons plus de détails pour l'insertion de données.

Interactions simples avec la base (4)

Insertion

```
DECLARE
    v_emp emp%ROWTYPE;
    v_nom emp.nome%TYPE;
BEGIN
    v_nom := 'Snoussi';
    insert into emp (matr, nome) values(600, v_nom);
    v_emp.matr := 610;
    v_emp.nome := ‘Rihani';
    insert into emp (matr, nome) values(v_emp.matr, v_emp.nome);
    commit;
END;
```

Interactions simples avec la base (5)

Autres exemples

```
declare
    v_emp emp%rowtype;
begin
    select * into v_emp from emp where nome = ‘Ghazi';
    v_emp.matr := v_emp.matr + 5;
    v_emp.nome := ‘Fahsi';
    insert into emp values v_emp;
end;
```

Curseurs

Définition

- **Curseur**: zone de mémoire de taille fixe, utilisée par le noyau d'Oracle pour analyser et interpréter tout ordre SQL.
- Deux types de curseurs :
  - **Implicite** : créés et gérés par Oracle à chaque ordre SQL (lorsque la close **INTO** accompagne le **SELECT).**
  - **Explicite :** créés et gérés par le programmeur afin de pouvoir traiter un **SELECT** qui retourne plusieurs tuples.
- Pour utiliser un curseur explicite, on doit passer par les étapes suivantes :
  1. **Déclaration** du curseur
  2. **Ouverture** du curseur
  3. **Traitement** des lignes du résultat
  4. **Fermeture** du curseur

Curseurs (2)

Déclaration du curseur :

- Association d'un nom de curseur à une requête **SELECT;**
- Se fait dans la section **DECLARE** d'un bloc PL/SQL;

```
DECLARE
  CURSOR nom_curseur IS Requête_Select ;
  ..........
```

➢ **Exemple** : employe (id, nomemp, sal, ville, …)

```
DECLARE
  CURSOR emp_rabat IS
  SELECT nomemp, sal FROM employe WHERE  ville = 'Rabat';
```

Curseurs (3)

Ouverture du curseur :

- Alloue un espace mémoire au curseur et positionne les éventuels verrous
- Après avoir déclaré le curseur, il faut l'ouvrir dans la section exécutable **BEGIN**.
- L'ouverture du curseur amorce l'**analyse** de **SELECT** et son exécution.
- La réponse est calculée et rangée dans un espace temporaire.

OPEN nom_curseur ;

- **Exemple** :

```
DECLARE
  CURSOR emp_rabat IS
  SELECT nomemp, sal FROM employe WHERE  ville = 'Rabat' ;
BEGIN
   OPEN emp_rabat ;
   ……
END ;
```

Curseurs (4)

Traitement des lignes

- L'accès aux données se fait par la clause : **FETCH INTO**

  FETCH nom_curseur **INTO** var1, var2, …
- **FETCH** permettant de récupérer une ligne de l'ensemble des lignes associés au curseur et de stocker les valeurs dans des variables réceptrices.
- Pour traiter plusieurs tuples, il faut utiliser une boucle.

Fermeture du curseur

Après le traitement des lignes pour libérer la place mémoire, il faut fermer le curseur: **CLOSE** nom_curseur ;

Curseurs (5)

Exemple 1 : employe (id, nomemp, sal, ville, …)

```
DECLARE
CURSOR emp_rabat IS SELECT nomemp, sal FROM employe
WHERE  ville = 'Rabat' ;
nom employe.nomemp%TYPE ;
salaire employe.sal%TYPE ;
BEGIN
OPEN emp_rabat ;
FETCH emp_rabat INTO nom, salaire; -- On récupère le premier enregistrement
WHILE emp_rabat%found LOOP -- S'il y a un enregistrement récupéré
            .....                    --Traitement de l'enregistrement récupéré
   FETCH emp_rabat INTO nom, salaire;-- On récupère l'enregistrement suivant
END  LOOP;
 CLOSE emp_rabat ;
END;
```

Curseurs (6)

Exemple 2 : création et remplissage d'une table '**resultat**' à partir de la table **employe**

```
CREATE TABLE resultat (nom varchar2(10), sal number(7,2)) ;
DECLARE
   CURSOR emp_rabat IS SELECT nomemp, sal FROM employe
            WHERE ville ='Rabat' ;
   nom employe.nomemp%TYPE ; salaire employe.sal%TYPE ;
BEGIN
   OPEN emp_rabat ;
   FETCH emp_rabat INTO nom, salaire
   WHILE emp_rabat%found LOOP
            IF salaire > 3000 THEN
                  INSERT INTO resultat VALUES (nom, salaire) ;
            END IF ;
           FETCH emp_rabat INTO nom, salaire
   END  LOOP;
   CLOSE emp_rabat ;
END ;
```

Curseurs (7)

Statut d'un curseur (Attribut)

- Les attributs d'un curseur sont des indicateurs sur son état.
- Quatre attributs permettent d'évaluer l'état du curseur:
  - ✓ **%Found** : vrai si le dernier **FETCH** a ramené un tuple.
  - ✓**%NotFound** : vrai si le dernier **FETCH** n'a ramené aucun tuple.
  - ✓**%Rowcount** : compte le nombre de **FETCH** exécutés sur un curseur;
  - ✓ **%Isopen** : vrai si le curseur est ouvert;

Curseurs (8)

Remarque :

Avec un curseur **implicite créé par** le SGBD Oracle, les mêmes attributs aux colonnes sont disponibles à condition de les préfixer par SQL. Ces attributs référent au **dernier** curseur implicite utilisé par l'application.

Curseur implicite	Curseur explicite
SQL%FOUND	nom-curseur**%FOUND**
SQL%NOTFOUND	nom-curseur**%NOTFOUND**
SQL%ISOPEN	nom-curseur**%ISOPEN**
SQL%ROWCOUNT	nom-curseur**%ROWCOUNT**

Curseurs (9)

Utilisation simplifiée des curseurs

- L'utilisation **FOR …. LOOP** remplace **OPEN**, **FETCH** et **CLOSE**.
- Lorsque le curseur est invoqué, un enregistrement est automatiquement créé avec les mêmes éléments de données que ceux définies dans **SELECT**.

Exemple: création et remplissage d'une table '**resultat**' à partir de la table **employe**

```
CREATE TABLE resultat (nom varchar2(10), sal number (7,2));
DECLARE
    CURSOR employe_rabat IS
    SELECT nomempl, sal FROM employe WHERE ville ='Rabat' ;
BEGIN
    FOR enrg_e IN employe_rabat LOOP
      IF enrg_e.sal >3000 THEN
       INSERT INTO resultat VALUES (enrg_e.nomempl, enrg_e.sal) ;
      END IF ;
    END LOOP ;
END ;
```

Curseurs (10)

Exercice

- Ecrire un bloc PL/SQL permettant de :
  - ✓ Créer deux tables **Empl1** et **Empl2** qui contiennent les colonnes **empno**, **ename**, **sal** et **deptno** de la table **Emp**.
  - ✓ Utiliser un curseur pour sélectionner les colonnes **empno**, **ename**, **sal** et **deptno** de tous les employés de la table **Emp**.
  - ✓ Parcourir ce curseur afin d'insérer dans **Empl1** les employés gagnant plus que 1500$ et dans **Empl2** les autres employés.
  - ✓ Afficher le nombre de tous les employés dans le curseur.

Curseurs (11)

L'attribut de curseur %ROWTYPE

- L'attribut **%ROWTYPE** permet de déclarer une variable de même type que l'enregistrement (la ligne) de la table.

  Syntaxe : Nom_de_variable nom_table**%ROWTYPE** ;

  Exemple 1:

  DECLARE

  enrg_empl empl%**ROWTYPE**

- Avec un curseur :

  CURSOR nom_curseur **IS** Requete_SELECT ;

  nom_variable nom_curseur**%ROWTYPE** ;

- Les éléments de la structure (nom_variable) sont identifiés par : nom_variable.nom_colonne
- La structure est renseignée par le **FETCH** :

  FETCH nom_curseur **INTO** nom_variable

Curseurs (12)

Exemple :

```
CREATE TABLE resultat (nom varchar2(10), sal number (7,2));
DECLARE
CURSOR emp_rabat IS SELECT nomemp, sal FROM  employe
   WHERE ville =‘Rabat’ ;
ligne emp_rabat%ROWTYPE ;
BEGIN
     OPEN emp_rabat ;
     FETCH emp_rabat INTO ligne ;
     WHILE emp_rabat%found LOOP
          IF ligne.sal >3000 THEN
             INSERT INTO resultat VALUES (ligne.nomemp, ligne.sal) ;
          END IF ;
          FETCH emp_rabat INTO ligne ;
     END LOOP ;
     CLOSE emp_rabat ;
END ;
```

Curseurs (13)

Utilisation d'une variable de type Record

- Un record permet de définir des types composites.

Syntaxe :

```
TYPE nom_record IS RECORD (
    v1 type1,
    v2 type2,
    ……  ) ;
```

- Déclaration d'une variable de ce type : nom_variable **nom_record**

```
DECLARE
    TYPE enrg_emp IS RECORD (
        nom    employe.nomempl %TYPE,
        salaire    employe.sal %TYPE ;
        e enrg_empl) ;
```

Curseurs (14)

Modification des données

La modification des données se fait habituellement avec **INSERT**, **UPDATE** ou **DELETE**, mais on peut utiliser: **FOR UPDATE** dans la déclaration du curseur.

Objectif du curseur modifiable:

- Modification via SQL sur le n-uplet courant
- Permet d'accéder directement en modification ou en suppression du nuplet récupéré par **FETCH**.

Remarque:

- Un curseur qui comprend **plus** d'une table dans sa définition ne permet pas la modification des tables de BD.
- Seuls les curseurs définis sur **une seule table sans fonction d'agrégation** et de regroupement peuvent utilisés dans une MAJ : delete, update, insert avec le **CURRENT OF CURSOR**.

Syntaxe:

CURSOR nomCurseur **IS ….. FOR UPDATE;**
... – déclarations des variables et opérations
…
WHERE CURRENT OF nomCurseur;
……

Curseurs (15)

Exemple :

```
DECLARE
  CURSOR ce IS
  SELECT nomempl, sal, ville FROM employe FOR UPDATE ;
  nom   employe.nomempl%TYPE ;
  salaire  employe.sal%TYPE ;
  ville  employe.ville%TYPE ;
BEGIN
  OPEN ce ;
  FETCH ce INTO nom, salaire, ville ;
  WHILE ce%found LOOP
        IF ce.ville IS NULL THEN
          DELETE FROM employe WHERE CURRENT OF ce ;
        END IF ;
        FETCH ce INTO nom, salaire, ville ;
  END LOOP ;
  CLOSE ce ;
END ;
```

Curseurs (16)

Exercice

- Vider la table Empl1.
- Ecrire un bloc PL/SQL permettant de :
  - ✓ Lire deux réel A et B avec A ≤ B
  - ✓ Utiliser un curseur pour sélectionner les colonnes **empno**, **ename**, **sal** et **deptno** des employés de la table **Emp** qui ont un salaire entre A et B.
  - ✓ Parcourir les lignes de ce curseur afin de les insérer dans **Empl1**.
  - ✓ Afficher le nombre de tous les employés dans le curseur.