

PL/SQL

Université de Nice Sophia-Antipolis
Version 0.5 – 11/12/04
Richard Grin

- ❑ Avertissement : cette partie du cours n'est qu'un survol du langage PL/SQL, utile pour écrire des procédures stockées simples
- ❑ Elle laisse de côté de nombreuses fonctionnalités du langage

Introduction

Pourquoi PL/SQL ?

- ❑ SQL est un langage non procédural
- ❑ Les traitements complexes sont parfois très difficile 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 de PL/SQL

- ❑ 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
- ❑ 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)

Normalisation du langage

- ❑ PL/SQL est un langage propriétaire de Oracle
- ❑ PostgreSQL utilise un langage très proche
- ❑ Ressemble au langage normalisé PSM (*Persistent Stored Modules*), mais pas plus que ça
- ❑ Cependant tous les langages des différents SGBD se ressemblent

Structure d'un programme

Blocs

- ❑ Un programme est structuré en blocs d'instructions de 3 types :
 - procédures anonymes
 - procédures nommées
 - fonctions nommées
- ❑ Les procédures anonymes ne peuvent être appelées ; elles sont placées au début de l'exécution

Structure d'un bloc

```
DECLARE
    -- définitions de variables
BEGIN
    -- Les instructions à exécuter
EXCEPTION
    -- La récupération des erreurs
END;
```

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

Blocs internes

- ❑ Un bloc peut contenir d'autres blocs

Les variables

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

Commentaires

- ❑ `--` Pour une ligne
- ❑ `/*` Pour plusieurs lignes `*/`

Types de variables

- ❑ Les types habituels correspondants aux types SQL2 ou Oracle : integer, varchar,...
- ❑ Types composites adaptés à la récupération des colonnes et lignes des tables SQL : %TYPE, %ROWTYPE
- ❑ Type référence : REF

Déclaration d'une variable

- ❑ `identificateur [CONSTANT] type [:= valeur];`
- ❑ Exemples :
 - `age integer;`
 - `nom varchar(30);`
 - `dateNaissance date;`
 - `ok boolean := true;`
- ❑ Déclarations multiples **interdites** :
~~`i, j integer;`~~

Déclaration %TYPE

- ❑ On peut déclarer qu'une variable est du même type qu'une colonne d'une table ou d'une vue (ou qu'une autre variable) :
`nom emp.nome.%TYPE;`

Déclaration %ROWTYPE

- ❑ Une variable peut contenir toutes les colonnes d'une ligne d'une table
- ❑ `employe emp%ROWTYPE;`
déclare que la variable employe contiendra des lignes de la table emp

Exemple d'utilisation

```
employe emp%ROWTYPE;  
select * INTO employe  
  from emp  
  where matr = 900;  
nom := employe.nom;  
employe.dept := 20;  
...  
insert into emp  
values employe;
```

Type RECORD

- Equivalent à struct du langage C
- TYPE nomRecord IS RECORD (
 champ1 type1,
 champ2 type2,
 ...);

Utilisation du type RECORD

```
TYPE emp2 IS RECORD (  
  matr integer,  
  nom varchar(30));  
employe emp2;  
employe.matr := 500;
```

Affectation

- Plusieurs façons de donner une valeur à une variable :
 - :=
 - par la directive INTO de la requête SELECT
- Exemples :
 - `dateNaissance := '10/10/2004';`
 - `select nome INTO nom
 from emp
 where matr = 509;`

Conflits de noms

- Si une variable porte le même nom qu'une colonne d'une table, c'est la colonne qui l'emporte
- DECLARE
 nom varchar(30) := 'DUPOND';
 BEGIN
 delete from emp where nom = nom;
- Pour éviter ça, le plus simple est de ne pas donner de nom de colonne à une variable !

Structures de contrôle

Alternative

- IF condition THEN
instructions;
END IF;
- IF condition1 THEN
instructions1;
ELSEIF condition2 THEN
instructions2;
ELSEIF ...
...
ELSE
instructionsN;
END IF;

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 par exemple pas être un BLOB ou un varray)

Boucle « tant que »

- WHILE condition LOOP
instructions;
END LOOP;

Boucle « jusqu'à »

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

Boucle « pour »

- FOR compteur IN [REVERSE] inf..sup LOOP
instructions;
END LOOP;
- Exemple :
for i IN 1..100 LOOP
somme := somme + i;
END LOOP;

Mise au point

- Pour la mise au point il est utile de faire afficher les valeurs des variables
- On peut le faire en activant sous SQL*PLUS la sortie sur l'écran et en utilisant le paquetage DBMS_OUTPUT
- Un paquetage est un regroupement de procédures et de fonctions ; notion pas vue dans ce cours

Exemple

```
set serveroutput on -- sous SQLPLUS
declare
  nb integer;
begin
  delete from emp
    where matr in (600, 610);
  nb := sql%rowcount; -- curseur sql
  dbms_output.put_line('nb = ' || nb);
end;
```

Interactions simples avec la base

Extraire des données

- ❑ On utilise « select into » pour récupérer la valeur d'une ou plusieurs colonnes dans une ou plusieurs variables
- ❑ Le select ne doit renvoyer qu'une seule ligne
- ❑ Avec Oracle il n'est pas possible d'inclure un select sans « into » dans une procédure ; pour ramener des lignes, voir la partie du cours sur les curseurs

Extraire des données – erreurs

- ❑ Si le select renvoie plus d'une ligne, une exception « TOO_MANY_ROWS » (ORA-01422) est levée (voir exceptions plus loin dans ce cours)
- ❑ Si le select ne renvoie aucune ligne, une exception « NO_DATA_FOUND » (ORA-01403) est levée

Exemple

```
DECLARE
  nom emp.nome%TYPE;
  v_emp emp%ROWTYPE;
BEGIN
  select nome into nom
  from emp
  where matr = 500;
  select * into v_emp
  from emp
  where matr = 500;
```

Récupérer plusieurs lignes

- ❑ Il faut utiliser des curseurs explicites (étudiés plus loin)

Modification de données

- ❑ On peut utiliser les variables dans les requêtes SQL (insert, update, delete,...)
- ❑ Si la variable est de type %ROWTYPE, il faut le plus souvent passer par les composants de la variable
- ❑ 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

Insertions

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

Autre exemple

```
declare
  v_emp emp%rowtype;
begin
  select * into v_emp from emp
    where nome = 'LEROY';
  insert into emp2
    values v_emp;
end;
```

Curseurs

Fonctionnalités

- ❑ Toutes les requêtes SQL sont associée à un curseur
- ❑ Ce curseur représente la zone mémoire utilisée pour parser et exécuter la requête
- ❑ Le curseur peut être implicite (pas déclaré par l'utilisateur) ou explicite
- ❑ Les curseurs explicites servent à retourner plusieurs lignes avec un select

Attributs des curseurs

- ❑ Tous les curseurs ont des attributs que l'utilisateur peut utiliser
 - %ROWCOUNT : nombre de lignes traitées par la requête
 - %FOUND : vrai si au moins une ligne a été traitée
 - %NOTFOUND : vrai si aucune ligne n'a été traitée
 - %ISOPEN : vrai si le curseur est ouvert (utile seulement pour les curseurs explicites)

Curseur implicite

- ❑ Les curseurs implicites sont tous nommés SQL

Exemple de curseur implicite

```
DECLARE
  nb_lignes integer;
BEGIN
  delete from emp
    where dept = 10;
  nb_lignes := SQL%ROWCOUNT;
  . . .
```

Curseur explicite

- ❑ Pour traiter les select qui renvoient plusieurs lignes
- ❑ Ils doivent être déclarés
- ❑ Le code doit les utiliser explicitement avec les ordres OPEN, FETCH et CLOSE
- ❑ Le plus souvent on les utilise dans une boucle dont on sort quand l'attribut NOTFOUND du curseur est vrai

Exemple (déclaration)

```
DECLARE
  CURSOR salaires IS
    select sal
    from emp
    where dept = 10;
  salaire numeric(8, 2);
  total numeric(8, 2) := 0;
```

Exemple (corps du bloc)

```
BEGIN
  open salaires;
  loop
    fetch salaires into salaire;
    if salaire is not null then
      total := total + salaire;
      DBMS_OUTPUT.put_line(total);
    end if;
    exit when salaires%notfound;
  end loop;
  close salaires; -- Ne pas oublier
  DBMS_OUTPUT.put_line(total);
END;
```

Attention !

Type « row » associé à un curseur

- ❑ On peut déclarer un type « row » associé à un curseur

- ❑ Exemple :

```
declare
  cursor c is
    select matr, nome, sal from emp;
  employe c%ROWTYPE;
begin
  open c;
  fetch c into employe;
  if employe.sal is null then ...
```

Boucle FOR pour un curseur

- ❑ Elle simplifie la programmation car elle évite d'utiliser explicitement les instructions open, fetch, close
- ❑ En plus elle déclare implicitement une variable de type « row » associée au curseur

Exemple

```
declare
  cursor c is
    select dept, nome from emp
    where dept = 10;
begin
  FOR employe IN c LOOP
    dbms_output.put_line(employe.nome);
  END LOOP;
end;
```

Curseur paramétré

- ❑ Un curseur paramétré peut servir plusieurs fois avec des valeurs des paramètres différentes
- ❑ On doit fermer le curseur entre chaque utilisation de paramètres différents (sauf si on utilise « for » qui ferme automatiquement le curseur)

Exemple

```
declare
  cursor c(p_dept integer) is
    select dept, nome from emp
    where dept = p_dept;
begin
  for employe in c(10) loop
    dbms_output.put_line(employe.nome);
  end loop;

  for employe in c(20) loop
    dbms_output.put_line(employe.nome);
  end loop;
end;
```

Ligne courante d'un curseur

- ❑ La ligne courante d'un curseur est déplacée à chaque appel de l'instruction fetch
- ❑ On est parfois amené à modifier la ligne courante pendant le parcours du curseur
- ❑ Pour cela on peut utiliser la clause « where current of » pour désigner cette ligne courante dans un ordre LMD (insert, update, delete)
- ❑ Il est nécessaire d'avoir déclaré le curseur avec la clause FOR UPDATE OF pour que le bloc compile

Exemple

```
DECLARE
  CURSOR c IS
    select matr, nome, sal
    from emp
    where dept = 10
    FOR UPDATE OF emp.sal;
...
  if salaire is not null then
    total := total + salaire;
  else -- on met 0 à la place de null pour le salaire
    update emp set sal = 0
    where current of c;
  end if;
```

Fonction qui renvoie un curseur sous Oracle

- ❑ Question : comment écrire une fonction (ou une procédure) qui renvoie un curseur ?
- 1. Créer un type pour la référence de curseur qu'on va renvoyer
- 2. Créer la fonction qui renvoie la référence de curseur
- ❑ Attention, solution propriétaire d'Oracle !

Créer le type référence de curseur

- ❑ Pour utiliser ensuite le type, il faut le créer dans un paquetage :

```
create or replace package Types AS
  type curseur_type is ref cursor;
end Types;
```

Créer la fonction

```
create or replace
function listdept(num integer)
  return Types.curseur_type
is
  empcursor Types.curseur_type;
begin
  open empcurseur for
  select dept, nomE
  from emp where dept = num;
  return empcurseur;
end;
```

Utiliser la fonction dans JDBC

```
CallableStatement cstmt =
  conn.prepareCall("{ ? = call list(?) }");
cstmt.setInt(2, 10);
cstmt.registerOutParameter(1,
  OracleTypes.CURSOR);
cstmt.execute();
ResultSet rs =
  ((OracleCallableStatement)cstmt)
  .getCursor(1);
while (rs.next()) {
  System.out.println(rs.getString("nomE")
    + ";" + rs.getInt("dept"));
}
```

Ne marche que sous Oracle !

Exceptions

Présentation

- ❑ Une exception est une erreur qui survient durant une exécution
- ❑ 2 types d'exception :
 - prédéfinie par Oracle
 - définie par le programmeur
- ❑ Une exception ne provoque pas nécessairement l'arrêt du programme si elle est saisie par un bloc (dans la partie « EXCEPTION »)

Exceptions prédéfinies

- ❑ NO_DATA_FOUND
- ❑ TOO_MANY_ROWS
- ❑ VALUE_ERROR (erreur arithmétique)
- ❑ ZERO_DIVIDE
- ❑ ...

Traitement des exceptions

```
BEGIN
...
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    . . .
  WHEN TOO_MANY_ROWS THEN
    . . .
  WHEN OTHERS THEN -- optionnel
    . . .
END;
```

On peut utiliser les 2 variables prédéfinies `SQLCODE` et `SQLERRM` (message)

Exceptions utilisateur

- ❑ Elles doivent être déclarées avec le type `EXCEPTION`
- ❑ On les lève avec l'instruction `RAISE`

Exemple

```
DECLARE
  salaire numeric(8,2);
  salaire_trop_bas EXCEPTION;
BEGIN
  select sal into salaire from emp where matr = 50;
  if salaire < 300 then raise salaire_trop_bas;
  end if;
  -- suite du bloc
EXCEPTION
  WHEN salaire_trop_bas THEN . . .;
  WHEN OTHERS THEN
    dbms_output.put_line(SQLERRM);
END;
```

Procédures et fonctions

Bloc anonyme ou nommé

- ❑ Dans `SQL*PLUS` on peut exécuter directement un bloc PL/SQL en tapant sa définition (on dit qu'on exécute un bloc anonyme)
- ❑ Le plus souvent, on crée plutôt une procédure ou une fonction nommée pour réutiliser le code

Création d'une procédure

- ❑

```
create or replace
PROCEDURE(<liste params>) IS
  -- déclaration des variables
BEGIN
  -- code de la procédure
END;
```
- ❑ Pas de DECLARE ; les variables sont déclarées entre IS et BEGIN

Création d'une fonction

- ❑

```
create or replace
FUNCTION(<liste params>)
RETURN <type retour> IS
  -- déclaration des variables
BEGIN
  -- code de la procédure
END;
```

Passage des paramètres

- ❑ Dans la définition d'une procédure on indique le type de passage que l'on veut pour les paramètres :
 - IN pour le passage par valeur
 - IN OUT pour le passage par référence
 - OUT pour le passage par référence mais pour un paramètre dont la valeur n'est pas utilisée en entrée
- ❑ Pour les fonctions, seul le passage par valeur (IN) est autorisé

Fonctions

- ❑ Un fois compilées, les fonctions peuvent être utilisées dans les requêtes SQL

Exemple de fonction

```
create or replace
function euro_to_fr(somme IN REAL)
RETURN REAL IS
  taux constant real := 6.55957;
begin
  return somme * taux;
end;
```

Utilisation de la fonction

```
select nome, sal, euro_to_fr(sal)
from emp;
```

Exécution d'une procédure

- Sous SQL*PLUS on exécute une procédure PL/SQL avec la commande EXECUTE :
EXECUTE *nomProcédure*(*param1*, ...);