

# **PL/SQL**

**Université de Nice Sophia-Antipolis**

**Version 0.6.3 – 3/12/06**

**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 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 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)
- Il est aussi utilisé dans des outils Oracle, *Forms* et *Report* en particulier

# Normalisation du langage

- PL/SQL est un langage propriétaire de Oracle
- PostgreSQL utilise un langage très proche
- Ressemble au langage normalisé PSM  
*(Persistant Stored Modules)*
- Tous les langages L4G des différents SGBDs se ressemblent

# Structure d'un programme

# Blocs

- Un programme est structuré en blocs d'instructions de 3 types :
  - procédures anonymes
  - procédures nommées
  - fonctions nommées
- Un bloc peut contenir d'autres blocs

# Structure d'un bloc

**DECLARE**

-- définitions de variables

**BEGIN**

-- Les instructions à exécuter

**EXCEPTION**

-- La récupération des erreurs

**END ;**

Seuls BEGIN et END  
sont obligatoires

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

# 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 fin de 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 une ligne de la table emp

# Exemple d'utilisation

```
employe emp%ROWTYPE;
nom emp.nome-%TYPE;
select * INTO employe
  from emp
  where matr = 900;
nom := employe.nome;
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**

```
    nome varchar(30) := 'DUPOND' ;
```

```
BEGIN
```

```
    delete from emp where nome = nome ;
```

- 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 condition THEN**  
    instructions1;  
**ELSE**  
    instructions2;  
**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 pas par exemple être un RECORD)

# Boucle « tant que »

- WHILE condition LOOP  
instructions;  
END LOOP;

# Boucle générale

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

- `select expr1, expr2, ... into var1, var2,`  
met des valeurs de la BD 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 suite 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 support de cours)
- Si le select ne renvoie aucune ligne, une exception « **NO\_DATA\_FOUND** » (ORA-01403) est levée

# Exemple

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

# 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

# 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';
    v_emp.matr := v_emp.matr + 5;
    v_emp.nome := 'Toto';
    insert into emp
        values v_emp;
end;
```

# Curseurs

# Fonctionnalités

- Toutes les requêtes SQL sont associées 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 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)

# 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(10, 2) := 0;
```

# Exemple (corps du bloc)

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

Variable de type  
c%rowtype

# 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** pour que le bloc compile

# FOR UPDATE

- **FOR UPDATE [OF *col1, col2, ...*]**
- Cette clause bloque toute la ligne ou seulement les colonnes spécifiées
- Les autres transactions ne pourront modifier les valeurs tant que le curseur n'aura pas quitté cette ligne

# 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 -- met 0 à la place de null
        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
    empcurseur 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

# Rappel de la structure d'un bloc

**DECLARE**

    -- définitions de variables

**BEGIN**

    -- Les instructions à exécuter

**EXCEPTION**

    -- La récupération des erreurs

**END ;**

# Saisir une exception

- Une exception ne provoque pas nécessairement l'arrêt du programme si elle est saisie par un bloc (dans la partie « EXCEPTION »)
- Une exception non saisie remonte dans la procédure appelante (où elle peut être saisie)

# 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**

# 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

- Un bloc anonyme PL/SQL est un bloc « DECLARE – BEGIN – END » comme dans les exemples précédents
- Dans SQL\*PLUS on peut exécuter directement un bloc PL/SQL anonyme en tapant sa définition
- 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
- Si la procédure ne nécessite aucune déclaration, le code est précédé de « IS 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

# Compilation

- Sous SQL\*PLUS, il faut taper une dernière ligne contenant « / » pour compiler une procédure ou une fonction

# Utilisation des procédures et fonctions

- Les procédures et fonctions peuvent être utilisées dans d'autres procédures ou fonctions ou dans des blocs PL/SQL anonymes

# Fonctions

- Les fonctions peuvent aussi être utilisées dans les requêtes SQL

# Exemple de fonction

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

# Utilisation dans un bloc anonyme

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

# Utilisation dans une requête SQL

```
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*,    );

# Bibliographie

- Pour cette introduction rapide je me suis inspiré du livre suivant :  
SQL pour Oracle de Soutou et Teste  
(Eyrolles)