PL/SQL - 2

Objectifs du cours

- Opérateurs
- Conversions de types de données
- Blocks imbriqués et portée des variables
- Structures de contrôle
- Interactions avec le serveur Oracle

Opérateurs

- Arithmétique
  +, -, *, /, **
- Concaténation
  ||
- Parenthèse pour contrôler
  les priorités des opérations
  (,)
- Affectation
  :=
- Comparaison
  =, <>, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN
- Logique
  AND, OR, NOT
- Conversion de types

Conversions de types de données

- Les mélanges de types
  - provoquent des erreurs
  - affectent les performances
- Fonctions de conversion :
  - TO_CHAR
  - TO_DATE
  - TO_NUMBER

- Exemple :

```
commentaire_v := USER || ‘: ’ || SYSDATE ; -- Types incompatibles
commentaire_v := USER || ‘: ’ || TO_CHAR(SYSDATE) ; -- ok
```

Opérateurs, exemples typiques

- Incrémenter l'index d'une boucle
  compte_v := compte_v + 1;
- Affectation de la valeur d'un drapeau booléen
  egales_v := (n1_v = n2_v);
- Rechercher si une variable a une valeur
  affecte_v := (nemp_v IS NOT NULL)

Blocks imbriqués et portée des variables (exercice)

```
DECLARE
    poids_v NUMBER(3) := 600 ;
    message_v VARCHAR2(255) := ‘Produit 10012’ ;
BEGIN
    DECLARE
        poids_v NUMBER(3) := 1 ;
        message_v VARCHAR2(255) := ‘Produit 11001’ ;
        pays_v VARCHAR2(50) := ‘Europe’ ;
    BEGIN
        poids_v := poids_v +1 ;
        pays_v := ‘Ouest-’ || pays_v ;
    END ;
    poids_v := poids_v +1 ;
    message_v := mesage_v || ‘ est en stock’ ;
     pays_v := ‘Ouest-’ || pays_v ;
END ;
```

Blocks imbriqués et portée des variables (règles)

- Un block peut toujours imbriquer un ou plusieurs autres blocks
- La portée d'une variable est limitée au block dans lequel la variable est définie
- Les variables définies dans un block sont visibles depuis les blocks qui y sont imbriqués
- Un block peut surcharger les variables définies dans le (les) block(s) dans lequel (lesquels) il est imbriqué
  => (1) dans le block le plus à l'intérieur la valeur affectée à la variable dans le block exterieur est perdue
  => (2) à la fin du block intérieur la valeur de la variable du block exterieur est récupérée

Blocks imbriqués et portée des variables

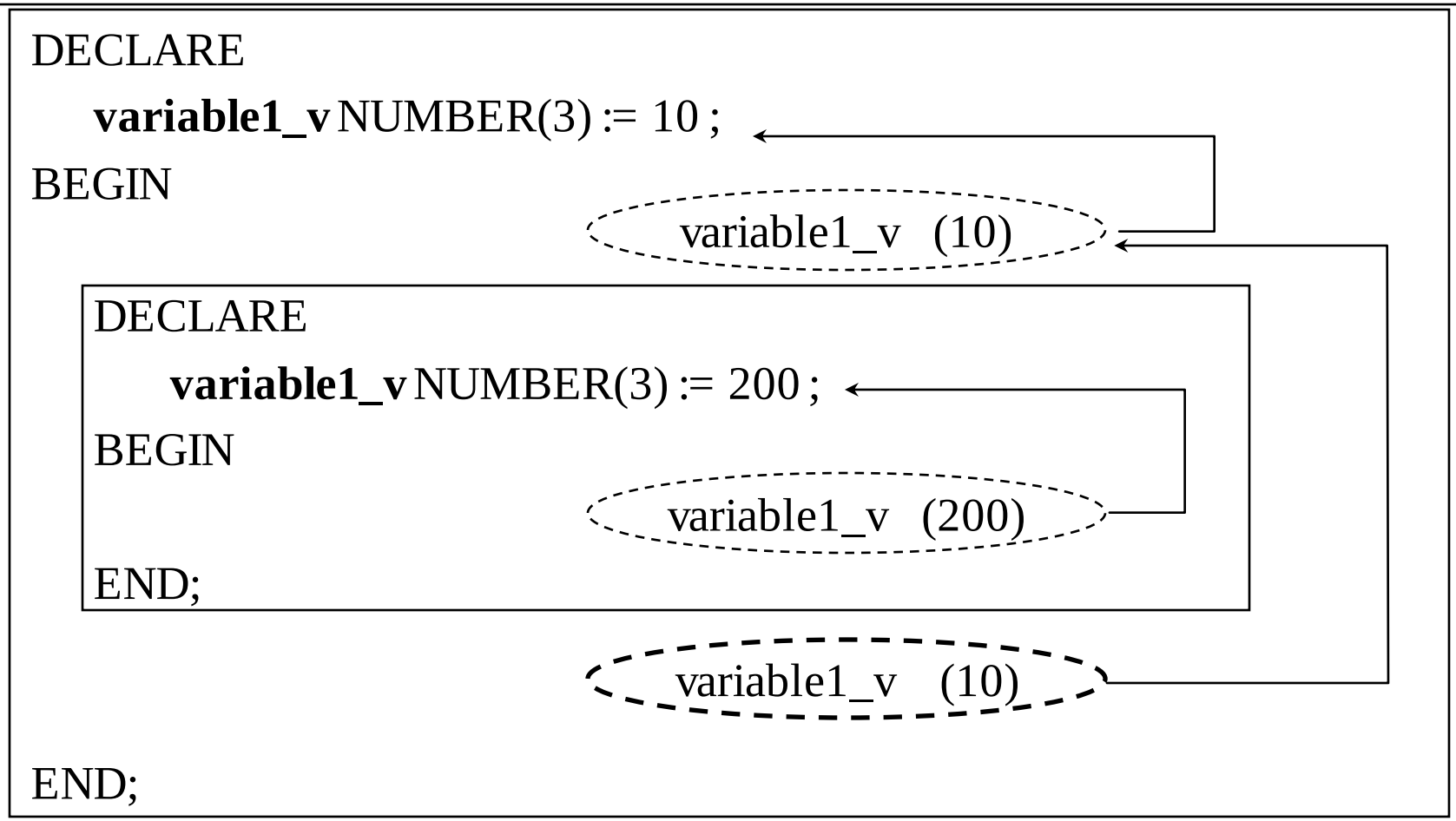
```
DECLARE
    variable1_v NUMBER(3) := 10 ;
BEGIN
    DECLARE
        variable2_v NUMBER(3) := 10 ;
    BEGIN
    END;
END;
```

variable1_v

variable1_v variable2_v

variable1_v

Blocks imbriqués et portée des variables



Blocks imbriqués et portée des variables

```
DECLARE
    poids_v NUMBER(3) := 600 ;
    message_v VARCHAR2(255) := 'Produit 10012' ;
BEGIN
    DECLARE
        poids_v NUMBER(3) := 1 ;                    -- (poids_v =1)
        message_v VARCHAR2(255) := 'Produit 11001' ;
        pays_v VARCHAR2(50) := 'Europe' ;
    BEGIN
        poids_v := poids_v +1 ;                     -- (poids_v = 2)
        pays_v := 'Ouest-' || pays_v ;
    END ;
    poids_v := poids_v +1 ;                         -- (poids_v  = 601)
    message_v := mesage_v || ' est en stock' ;      -- (message_v = 'Produit 10012 est en stock')
    pays_v := 'Ouest-' || pays_v ;                  -- ILLEGAL
END ;
```

Structures de contrôle

- Branchements conditionnels
  - IF - THEN - END IF
  - IF - THEN - ELSE - END IF
  - IF - THEN - ELSIF - END IF
- Boucles
  - LOOP - END LOOP
  - FOR - END LOOP
  - WHILE - END LOOP

  - Note : la commande EXIT permet de sortir de tout type de boucle

Structures de contrôle - branchements conditionnels

- Syntaxe :

IF <condition> THEN
    commandes ;
[ELSIF <condition> THEN
    commandes ;]*
[ELSE
    commandes ;]
END IF ;

Note :
vous pouvez utiliser l'expression IS NULL dans les conditions

- Exemples :

IF nomEmploye_v = 'TOTO' THEN
    salaire_v := salaire_v * 2 ;
END IF;

Structures de contrôle - branchements conditionnels

```
IF nomEmploye_v = 'TOTO' THEN
    salaire_v := salaire_v * 2 ;
ELSE
     salaire_v := salaire_v * 3 ;
END IF;
```

```
IF nomEmploye_v = 'TOTO' THEN
    salaire_v := salaire_v * 2 ;
ELSIF salaire_v > 10000 THEN
    salaire_v := salaire_v / 2 ;
ELSE
     salaire_v := salaire_v * 3 ;
END IF;
```

Structures de contrôle - boucles LOOP

- Note :
  Sans commande EXIT, les boucles LOOP sont infinies
- Syntaxe :

```
LOOP
    commandes ;
    . . .
    EXIT [WHEN <condition>];
END LOOP ;
```

Structures de contrôle - boucles LOOP

- Exemple :

```
DECLARE
    noEmp_v NUMBER (3) := 1;
BEGIN
    LOOP
        INSERT      INTO Employe (noEmp, nomEmp, job, noDept)
                    VALUES (noEmp_v, 'Oracle', 'PROGRAMMEUR',
    10) ;
        noEmp_v := noEmp_v +1 ;
        EXIT WHEN noEmp > 100 ;
    END LOOP ;
END ;
```

Structures de contrôle - boucles FOR

- Syntaxe :

  FOR <compteur> IN [REVERSE] <limite_inf> .. <limite_sup>
  commandes ;
  . . .
  END LOOP

- Exemple :

```
DECLARE
    noEmp_v NUMBER (3);
BEGIN
    FOR noEmp_v IN 1 .. 100
        INSERT      INTO Employe (noEmp, nomEmp, job, noDept)
                    VALUES (noEmp_v, 'Oracle', 'PROGRAMMEUR', 10) ;
    END LOOP ;
END ;
```

Structures de contrôle - boucles WHILE

- Syntaxe :

```
WHILE <condition> LOOP
    commandes ;
    . . .
END LOOP ;
```

- Exemple :

```
DECLARE
    noEmp_v NUMBER (3);
BEGIN
    noEmp_v := 1;
    WHILE noEmp_v <= 100 LOOP
        INSERT      INTO Employe (noEmp, nomEmp, job, noDept)
                    VALUES (noEmp_v, 'Oracle', 'PROGRAMMEUR', 10) ;
        noEmp_v := noEmp_v +1 ;
    END LOOP ;
END ;
```

Structures de contrôles - remarques sur les boucles

- Ne pas modifier le compteur d'une boucle FOR
- Les boucles peuvent être imbriquées
- On peut nommer les boucles pour identifier explicitement laquelle de deux boucles imbriquées se termine

```
. . .
<<boucleExterne>>
LOOP
      . . .
      EXIT WHEN compteur_v = 10 ;
      <<boucleInterne>>
      LOOP
            EXIT boucleExterne WHEN compteur_v = 100 ;
            EXIT boucleInterne WHEN drapeau_v = TRUE ;
      END LOOP boucleInterne ;
END LOOP boucleExterne ;
. . .
```

Interactions avec le serveur Oracle

- Inclure une requête SELECT dans un block PL/SQL
- Déclarer dynamiquement des variables de type adapté au SELECT
- Modifier des données dans PL/SQL
- Contrôler les transactions dans PL/SQL
- Déterminer le résultat d'une requête SELECT dans PL/SQL

Inclure une requête SELECT dans PL/SQL

```
DECLARE
    noDept_v NUMBER(2) ;
    lieu_v VARCHAR2(15) ;
BEGIN
    SELECT    noDept, lieu
        INTO  noDept_v, lieu_v
    FROM      Departement
    WHERE     nomDept = 'VENTES' ;
. . .
END ;
```

ATTENTION *:*

la requête ne doit retourner que un et un seul tuple !!!

Si ce n'est pas le cas, les exceptions NO_DATA_FOUND ou TOO_MANY_ROWS sont levées

Déclarer dynamiquement des variables de type adapté au SELECT

- %TYPE
  identifie dynamiquement le type d'un attribut d'une table
- %ROWTYPE
  identifie dynamiquement le type (structuré) d'un tuple d'une table

- Exemples :
  -- Commande est une table de la base

```
DECLARE
    dateCommande_v Commande.dateCommande%TYPE ;
    uneCommande_v Commande%ROWTYPE ;
. . .
```

Exemple de requête SELECT dans PL/SQL

```
DECLARE
        noDept_v Departement.noDept%TYPE;
        lieu_v Departement.lieu%TYPE;
BEGIN
        SELECT      noDept, lieu
                INTO   noDept_v, lieu_v
        FROM Departement
        WHERE       nomDept = 'VENTES';
. . .
END ;
```

Exemple de requête SELECT dans PL/SQL

```
DECLARE
    sommeSalaires_v Employe.salaire%TYPE ;
    noDept_v NUMBER NOT NULL := 10 ;
BEGIN
    SELECT   SUM(salaire)
        INTO  sommeSalaires_v
    FROM      Employe
    WHERE    noDepartement = noDept_v ;
END ;
```

Modifier des données dans PL/SQL

- Trois commandes du langage de manipulation de données (LMD) de SQL permettent de modifier une base de données :
  - INSERT
  - UPDATE
  - DELETE

Modifier des données dans PL/SQL (INSERT)

```
BEGIN
    INSERT    INTO Employe (noEmp, nomEmp, job, noDept)
              VALUES (10, 'Oracle', 'PROGRAMMEUR', 10) ;
END ;
```

- Note :
  - on peut évidemment utiliser des variables au lieu de simples valeurs prédéfinies
  - il peut être utile d'utiliser des variables globales prédéfinies comme USER ou SYSDATE

Modifier des données dans PL/SQL (UPDATE)

```
DECLARE
    majorationSalaire_v Employe.salaire%TYPE := 2000;
BEGIN
    UPDATE   Employe
    SET      salaire = salaire + majorationSalaire_v
    WHERE    job = 'PROGRAMMEUR';
END ;
```

- Note :
  - contrairement aux affectations PL/SQL, la clause update utilise le signe '=' comme opérateur d'affectation
  - si une variable a le même nom qu'un nom d'attribut de la table manipulée dans la clause WHERE, le serveur Oracle utilise en priorité l'attribut de table

Modifier des données dans PL/SQL (DELETE)

```
DECLARE
    noDept_v Employe.noDept%TYPE := 10 ;
BEGIN
    DELETE   FROM Employe
                 WHERE noDept = noDept_v ;
END ;
```

Contrôler les transactions dans PL/SQL

- La première commande INSERT/UPDATE/DELETE/CREATE/DROP d'un block entamme une nouvelle transaction
- La fin du block **ne termine pas** la transaction
- Pour terminer explicitement une transaction, utiliser les commandes SQL :
  - COMMIT
    => valide les modifications faites depuis le début de la transaction en cours, et entamme une nouvelle transaction
  - ROLLBACK
    => annule toutes les modifications faites depuis le début de la transaction en cours , et entamme une nouvelle transaction

- Note :
  Une transaction doit être un ensemble homogène de manipulations de la base de données => il faut réflechir à tous les endroits où il est légitime de mettre un COMMIT

Contrôler les transactions dans PL/SQL

```
DECLARE
    noDept_v Employe.noDept%TYPE := 10 ;
    majorationSalaire_v Employe.salaire%TYPE := 2000;
BEGIN
    DELETE   FROM Employe
             WHERE noDept = noDept_v ;
    COMMIT ;
    UPDATE   Employe
    SET      salaire = salaire + majorationSalaire_v
    WHERE    job = 'PROGRAMMEUR' ;
END ;
```

Déterminer le résultat d'une requête SELECT dans PL/SQL

- Nécessite l'utilisation de **curseurs**
  les curseurs sont des zones de travail privées
- Il y a deux types de curseurs :
  - les curseurs implicites
    le serveur Oracle utilise des curseurs implicites pour exécuter vos requêtes SQL
  - les curseurs explicites
    sont des variables explictement déclarées par le programmeur

Déterminer le résultat d'une requête dans PL/SQL

- Attributs des curseurs
  en utilisant les attributs de curseurs, vous pouvez tester le résultat de vos requêtes SQL
  - SQL%ROWCOUNT
    nombre de tuples affectés par la dernière requête SQL (entier)
  - SQL%FOUND
    bolléen, vaut TRUE si la dernière requête SQL a affecté au moins un tuple
  - SQL%NOTFOUND
    bolléen, vaut TRUE si la dernière requête SQL n'a affecté aucun tuple
  - SQL%ISOPEN
    boléen indiquant si le curseur est ouvert ou fermé (par défaut ,les curseurs implicites sont toujours fermés à la fin de la requête)

- Note :
  à la place de 'SQL', utilisez le nom de votre curseur pour identifier l'état d'un curseur explicite

Déterminer le résultat d'une requête dans PL/SQL

- Exemple : supprimer des tuples de la table Employe, imprimer le nombre de tuples supprimés

```
DECLARE
    nbreTuplesSupprimes VARCHAR2(20);
    noDept_v Employe.noDept%TYPE := 10 ;
BEGIN
    DELETE    FROM Employe
                  WHERE noDept = noDept_v ;
     nbreTuplesSupprimes := TO_CHAR(SQL%ROWCOUNT) || ' tuples
    supprimés' ;
END ;
```