



Burkina Faso

Unité- Progrès-Justice

**ORACLE DATABASE : SQL/SQL\*PLUS → 9 h**  
**ORACLE DEVELOPER : PL/SQL → 8 h**  
**Devoir 1 : → 2 Heures ( Mercredi 05 Mai 2021)**  
**ORACLE DEVELOPER : FORMS & REPORTS → 9 h**  
**Devoir 2 : → 2 Heures**

Durée : 30 Heures

Enseignante :  
**Madame IMA RASMATA**  
Email : ima\_ramatou@yahoo.fr  
Tel : 00226 70 73 38 41  
Whatsapp : 00226 77 55 96 62

Période :  
**Avril - Mai 2021**



Burkina Faso  
Unité- Progrès-Justice

## ORACLE DEVELOPER : ORACLE PL/SQL et l'outil SQL\*PLUS

Durée : 8 Heures

Enseignante :

**Madame IMA RASMATA**

Période :

**Avril - Mai 2021**

# PROGRAMME

## 0. INSTALLATION DE ORACLE DATABASE VERSION 11G , ET ACCÈS À L'OUTIL SQL\*PLUS

### 1. INTRODUCTION à PL\*SQL

### 2. STRUCTURE D'UN BLOC PL/SQL

### 3. LES VARIABLES UTILISÉES DANS PL/SQL

#### 3.1. LES DIFFÉRENTS TYPES DE VARIABLES LOCALES

3.1.1. Variables de type ORACLE

3.1.2. Variables de type BOOLEEN 1

3.1.3. Variables faisant référence au dictionnaire de données

3.1.4. Initialisation des variables

3.1.5. Visibilité des variables

#### 3.2. VARIABLES DE L'ENVIRONNEMENT EXTÉRIEUR À PL/SQL

### 4. LES TRAITEMENTS

4.1. IF : TRAITEMENT CONDITIONNEL

4.2. BOUCLE DE BASE LOOP : TRAITEMENT RÉPÉTITIF

4.3. BOUCLE FOR : TRAITEMENT RÉPÉTITIF

4.4. BOUCLE WHILE : TRAITEMENT RÉPÉTITIF

# PROGRAMME

## 5. LES CURSEURS EN PL/SQL

### 5.1. DÉFINITIONS

### 5.2. CURSEUR EXPLICITE

### 5.3. LES ATTRIBUTS D'UN CURSEUR

#### 5.3.1. %FOUND

#### 5.3.2. %NOTFOUND

#### 5.3.3. %ISOPEN

#### 5.3.4. %ROWCOUNT

### 5.4. SIMPLIFICATION D'ÉCRITURE

#### 5.4.1. Déclaration de variables

#### 5.4.2. Traitement du curseur

## 6. GESTION DES ERREURS EN PL/SQL

## 7. EXERCICES PL/SQL

### 7.1. EX1 : LES BOUCLES

### 7.2. EX2 : LES CURSEURS

### 7.3. EX3 : LES ERREURS

## 1. INTRODUCTION

**SQL** : est un langage ensembliste et non procédural

**PL/SQL** : est un langage procédural qui intègre des ordres SQL de gestion de la base de données

Instructions SQL intégrées dans PL/SQL :

- SELECT
- INSERT, UPDATE, DELETE
- COMMIT, ROLLBACK, SAVEPOINT
- TO\_CHAR, TO\_DATE, UPPER, ...

Instructions spécifiques à PL/SQL :

- définition de variables
- traitements conditionnels
- traitements répétitifs
- traitement des curseurs
- traitement des erreurs

## Structure d'un bloc PL/SQL

Un bloc PL/SQL est divisé en 3 sections :

DECLARE

Déclaration de variables, constantes,  
exceptions, curseurs

BEGIN [nom\_du\_bloc]

Instructions SQL et PL/SQL

EXCEPTION

Traitement des exceptions  
(gestion des erreurs)

END [nom\_du\_bloc] ;

Remarques :

- les sections DECLARE et EXCEPTION sont facultatives.
- chaque instruction se termine par un ;
- Les commentaires :
  - sur une ligne
  - ou
  - /\* sur plusieurs
  - lignes \*/

## 2. Structure d'un bloc PL/SQL (suite ...)

### Exemple :

PROMPT nom du produit

SQL

ACCEPT prod

DECLARE

PL/SQL

qte NUMBER(5)

BEGIN

SELECT quantite INTO qte

FROM stock

WHERE produit= '&prod';

-- on contrôle le stock

IF qte > 0 THEN

UPDATE stock

SET quantite = quantite - 1

WHERE produit = '&prod';

INSERT INTO vente

VALUES('&prod' || 'VENDU' , SYSDATE);

ELSE INSERT INTO commande

VALUES('&prod' || 'DEMANDE' , SYSDATE);

END IF;

COMMIT;

END;

/

### 3. Les variables utilisées dans PL/SQL

#### 3.1. Les différents types de variables locales

Les variables locales se déclarent dans la partie DECLARE du bloc PL/SQL.

Différents types de variables :

- \* Variables de types ORACLE
- \* Variables de type BOOLEAN
- \* Variables faisant référence au dictionnaire de données

Initialisation des variables

Visibilité des variables



### 3.1.2. Variables de type BOOLEAN

Syntaxe :

```
nom_var BOOLEAN;
```

Exemple :

```
DECLARE  
    retour          BOOLEAN;  
BEGIN  
    ...  
END;
```

### 3.1.3. Variables faisant référence au dictionnaire de données

\* Variable de même type qu'un attribut d'une table de la base

Syntaxe :

```
nom_var    table.colonne%TYPE;
```

Exemple :

```
DECLARE
    nom pilote.plnom%TYPE;
BEGIN
    ...
END;
```

### 3.1.3. Variables faisant référence au dictionnaire de données (suite ...)

- Variable de même structure qu'une ligne d'une table de la base

Syntaxe :

```
nom_var  table%ROWTYPE;
```

Exemple :

```
DECLARE
    ligne pilote%ROWTYPE;
BEGIN
    ...
END;
```

**Remarque :**

La structure ligne contient autant de variables que de colonnes de la table. Ces variables portent le même nom et sont de même type que les colonnes de la table.

Pour y accéder :

```
ligne.<nom_col1>
ligne.<nom_col2>
...
ligne.<nom_coln>
```

### 3.1.3. Variables faisant référence au dictionnaire de données (suite ...)

- Variable de même type qu'une autre variable

Syntaxe :

```
nom_var2  nom_var1%TYPE;
```

Exemple :

```
DECLARE
    ancien_sal    NUMBER(5);
    nouveau_sal   ancien_sal%TYPE;--NUMBER(5);
```

```
BEGIN
```

```
    ...
```

```
END;
```

# TYPE DE DONNÉES:

Les types SQL (number, varchar2(n), date, char(n), ..)

- Le type Boolean
- Le type variable %TYPE
- Le type rangée %ROWTYPE
- Les types CURSOR. (dynamique ou non)
- Les types RECOR

# EXAMPLE: TYPE RECORD

```
SET SERVEROUTPUT ON;
DECLARE
TYPE ENRE IS RECORD
(
Nom1 USER1.ETUDIANTS.NOM%TYPE,
prenom1 VARCHAR2(20)
);
numad1 NUMBER:=20;
ENR ENRE;
BEGIN
SELECT nom, prenom INTO ENR.nom1,ENR.prenom1
FROM etudiants
WHERE numad = numad1;
DBMS_OUTPUT.PUT_LINE('le nom est ' || ENR.nom1 || 'le prenom est '
|| ENR.prenom1);
END;
```

### 3.1.4. Initialisation des variables

Avec :

opérateur :=

ou

SELECT ... INTO ...

Exemple :

```
DECLARE
  var1 CHAR(10)      := 'DUPONT';
  var2 NUMBER(5,2)    := 100;
  var3 CHAR(10);
  var4 DATE;
BEGIN
  SELECT col1, col2
  INTO var3, var4
  FROM ... ;
  ...
END;
```

Remarque :

le SELECT doit ramener une et une seule ligne, sinon erreur.

### 3.1.5. Visibilité des variables

Une variable est visible dans le bloc où elle a été déclarée et dans les blocs imbriqués si elle n'a pas été redéfinie.

```
DECLARE
```

```
    var1 NUMBER(3);
```

```
    var2 CHAR(10);
```

```
BEGIN
```

```
    ...
```

```
    ...
```

```
    DECLARE
```

```
        var1 CHAR(10);
```

```
        var3 DATE;
```

```
    BEGIN
```

```
        ...
```

```
        ...
```

```
        ...
```

```
    END;
```

```
    ...
```

```
    DECLARE
```

```
        var4 NUMBER(5,2);
```

```
    BEGIN
```

```
        ...
```

```
        ...
```

```
        ...
```

```
    END;
```

```
    ...
```

```
    ...
```

```
END;
```

```
var1 NUMBER(3)
```

```
var2 CHAR(10)
```

```
var1 CHAR(10)
```

```
var2
```

```
var3
```

```
var1 NUMBER(3)
```

```
var2
```

```
var4
```

```
var1 NUMBER(3)
```

```
var2 CHAR(10)
```



## 3.2. Variables de l'environnement extérieur à PL/SQL

Outre les variables locales vues précédemment, un bloc PL/SQL peut utiliser d'autres variables :

- les champs d'écrans FORMS,
- les variables définies en langage hôte (préfixée de :)
- les variables définies dans SQL\*Plus (préfixée de &)



## 4. Les traitements

### 4.1. IF : traitement conditionnel

Exécution d'un traitement en fonction d'une condition.

```
IF    condition1 THEN traitement 1;
```

```
ELSIF condition2 THEN traitement 2;
```

```
[ELSE traitement 3;]
```

```
END IF;
```

Les opérateurs utilisés dans les conditions sont les mêmes que dans SQL :

=, <, ... IS NULL, LIKE, ...

Dès que l'une des conditions est vraie, le traitement qui suit le THEN est exécuté.  
Si aucune condition n'est vraie, c'est le traitement qui suit le ELSE qui est exécuté.

# Exemple 1

```
Declare choix number;  
begin  
  IF choix =1 THEN  
    delete from commander  
    where numarticle = 100 ;  
  ELSIF choix =2 THEN  
    delete from commander  
    where numarticle = 110;  
  ELSE  
    delete from commander  
    where numarticle = 130;  
  END IF;  
END;
```

## Exemple 2

Exercice: écrire un bloc PL/SQL qui permet de déclarer deux variable de type number (vente et bonus) et qui met à jour le salaire de l'employé comme suit: (salaire = salaire+bonus)

- Si vente est > 1000 alors bonus = vente\*50%
- Sinon bonus = vente \*20%

# Réponse



```
CREATE OR REPLACE PROCEDURE Augmentation (vente  
in number) AS  
Bonus number;  
BEGIN  
    IF VENTE > 1000 THEN BONUS := VENTE*0.5;  
    ELSE  
        BONUS := VENTE*0.2;  
    END IF;  
    UPDATE EMPLOYES  
    SET SALAIRE = SALAIRE+BONUS;  
    Commit;  
END;
```

# CASE --- WHEN

L'instruction CASE: permet d'exécuter un bloc PL/SQL selon la valeur d'une variable

**Exemple:**

```
CREATE OR REPLACE PROCEDURE CASE1(CHOIX IN NUMBER) AS
BEGIN
    CASE CHOIX
        WHEN 1 THEN
            INSERT INTO employes (numemp, salaire )
            VALUES (44,28000);
        WHEN 2 THEN
            UPDATE employes
            SET salaire = salaire +10 where
            numemp =10;
        ELSE
            dbms_output.put_line('pas bon choix');
        END CASE;
    END;
```

## 4.2. Boucle de base LOOP : traitement répétitif

Exécution d'un traitement plusieurs fois, le nombre n'étant pas connu mais dépendant d'une condition.

```
BEGIN
    LOOP [label]
        instructions;
    END LOOP [label];
END;
```

Pour sortir de la boucle, utiliser la clause :  
EXIT [label] WHEN condition

Exemple : insérer les 10 premiers chiffres dans la table result

```
DECLARE
    nb    NUMBER := 1;
BEGIN
    LOOP
        INSERT INTO result
        VALUES (nb);
        nb := nb + 1;
        EXIT WHEN nb > 10;
    END LOOP;
END ;
```

## 4.2. Boucle de base LOOP : traitement répétitif

Exécution d'un traitement plusieurs fois, le nombre n'étant pas connu mais dépendant d'une condition.

```
BEGIN
  LOOP [label]
    instructions;
  END LOOP [label];
END;
```

Pour sortir de la boucle, utiliser la clause :  
EXIT [label] WHEN condition

Exemple : insérer les 10 premiers chiffres dans la table result

```
DECLARE
  nb  NUMBER := 1;
BEGIN
  LOOP
    INSERT INTO result
    VALUES (nb);
    nb := nb + 1;
    EXIT WHEN nb > 10;
  END LOOP;
END ;
```



## 4.2. Boucle de base LOOP : traitement répétitif

```
CREATE OR REPLACE FUNCTION COMPTER RETURN NUMBER
AS
compteur number:=0;
BEGIN
    LOOP
        compteur:=compteur+1;
        EXIT WHEN compteur=10;
    END LOOP ;
    RETURN compteur;
END;
```

### 4.3. Boucle FOR : traitement répétitif

Exécution d'un traitement un certain nombre de fois. Le nombre étant connu.

```
BEGIN
  FOR indice IN [REVERSE] exp1 ... exp2
  LOOP
    instructions;
  END LOOP;
```

END;

Remarques :

- inutile de déclarer indice
- indice varie de exp1 à exp2 de 1 en 1
- si REVERSE est précisé, indice varie de exp2 à exp1 avec un pas de -1.

Exemple : calcul de la factorielle 5

```
DECLARE
  fact NUMBER := 1;
BEGIN
  FOR i IN 1 .. 5
  LOOP
    fact := fact * i ;
  END LOOP;
END;
```

## 4.3. Boucle FOR : traitement répétitif

```
set serveroutput on;  
BEGIN  
FOR i IN 1..3 LOOP  
  DBMS_OUTPUT.PUT_LINE (TO_CHAR(i));  
END LOOP;  
END;
```



## 4.3. Boucle FOR : traitement répétitif : Décrémentation



Exemple

```
set serveroutput on;  
BEGIN  
    FOR i IN REVERSE 1..3 LOOP  
        DBMS_OUTPUT.PUT_LINE (TO_CHAR(i));  
    END LOOP;  
END;
```

#### 4.4. Boucle WHILE : traitement répétitif

Exécution d'un traitement tant qu'une condition reste vraie.

```
BEGIN
  WHILE condition
  LOOP
    instructions;
  END LOOP;
END;
```

Exemple : reste de la division de 5432 par 5

```
DECLARE
  reste NUMBER := 5432;
BEGIN
  WHILE reste >= 5
  LOOP
    reste := reste -5;
  END LOOP;
END;
```

## 4.4. Boucle WHILE : traitement répétitif

Exécution d'un traitement tant qu'une condition reste vraie.

```
BEGIN
  WHILE condition
  LOOP
    instructions;
  END LOOP;
END;
```

Exemple : reste de la division de 5432 par 5

```
DECLARE
  reste NUMBER := 5432;
BEGIN
  WHILE reste >= 5
  LOOP
    reste := reste -5;
  END LOOP;
END;
```

## 4.4. Boucle WHILE : traitement répétitif

```
set serveroutput on;  
DECLARE  
    I NUMBER:=1;  
BEGIN  
    WHILE I < 10 LOOP  
        I:= I+1;  
        DBMS_OUTPUT.PUT_LINE (TO_CHAR(I));  
    END LOOP;  
END;
```

## 5. Les curseurs en PL/SQL

### 5.1. Définitions

Il existe 2 types de curseurs :

- CURSEUR IMPLICITE :

curseur SQL généré et géré par le noyau pour chaque ordre SQL d'un bloc.

- CURSEUR EXPLICITE :

curseur SQL généré et géré par l'utilisateur pour traiter un ordre SELECT qui ramène plus d'une ligne.



## 5.2. Curseur explicite

4 étapes :

- déclaration du curseur
- ouverture du curseur
- traitement des lignes
- fermeture du curseur

## 5.2. Cursor : statique (explicite)

Un Curseur statique ou explicite est obtenu par l'exécution d'une commande SQL.

- Pour son utilisation, il faut quatre étapes:
  - Déclaration du curseur
  - Ouverture du curseur OPEN
  - Lecture du curseur FETCH
  - Fermeture du curseur CLOSE

## 5.2. Cursor : statique (explicite)

```
SET SERVEROUTPUT ON;
DECLARE
    Nom1 USER1.ETUDIANTS.NOM%TYPE;
    Prenom1 varchar2(20);
    CURSOR curseur1 IS SELECT nom, Prenom from etudiants;
BEGIN
    OPEN curseur1;
    LOOP
        FETCH curseur1 INTO Nom1, Prenom1;
        EXIT WHEN curseur1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('le nom est '|| nom1 || 'le
        prenom est '|| prenom1);
    END LOOP;
    CLOSE Curseur1;
END;
```

## 5.2. Curseur explicite (suite ...)

Déclaration du curseur

déclaration dans la section DECLARE du bloc.

on indique le nom du curseur et l'ordre SQL associé

Syntaxe :

```
CURSOR nom_curseur IS ordre_select ;
```

Exemple :

```
DECLARE
    CURSOR pl_nice IS
        SELECT pl#, plnom
        FROM   pilote
        WHERE  adr='Nice';
BEGIN
    ...
END ;
```

## 5.2. Curseur explicite (suite ...)

Déclaration du curseur  
déclaration dans la section DECLARE du bloc.  
on indique le nom du curseur et l'ordre SQL associé

Syntaxe :

```
CURSOR nom_curseur IS ordre_select ;
```

Exemple :

```
DECLARE
    CURSOR pl_nice IS
        SELECT pl#, plnom
        FROM   pilote
        WHERE  adr='Nice';
BEGIN
    ...
END ;
```

## Ouverture du curseur

L'ouverture du curseur lance l'exécution de l'ordre SELECT associé au curseur.  
Ouverture dans la section BEGIN du bloc.

Syntaxe :

```
OPEN nom_curseur ;
```

Exemple :

```
DECLARE
    CURSOR pl_nice IS
        SELECT pl#, plnom
        FROM   pilote
        WHERE  adr='Nice';
BEGIN
    ...
    OPEN pl_nice;
    ...
END ;
```

## Traitement des lignes

Après l'exécution du SELECT les lignes ramenées sont traitées une par une, la valeur de chaque colonne du SELECT doit être stockée dans une variable réceptrice.

Syntaxe :

```
FETCH nom_curseur INTO liste_variables ;
```

Exemple :

```
DECLARE
```

```
    CURSOR      pl_nice IS
        SELECT   pl#, plnom, sal
        FROM     pilote
        WHERE    adr='Nice';
        num      pilote.pl#%TYPE;
        nom      pilote.plnom%TYPE;
        salaire   pilote.sal%TYPE;
```

```
BEGIN
```

```
    OPEN pl_nice;
```

```
    LOOP
```

```
        FETCH pl_nice INTO num,
                               nom,salaire;
```

```
        ...
```

```
        EXIT WHEN sal > 10 000;
```

```
    END LOOP;
```

```
END ;
```

## Fermeture du curseur

Pour libérer la mémoire prise par le curseur, il faut le fermer dès qu'on n'en a plus besoin.

Syntaxe :

```
CLOSE nom_curseur ;
```

Exemple :

```
DECLARE
```

```
    CURSOR pl_nice IS  
        SELECT pl#, plnom, sal  
        FROM   pilote  
        WHERE  adr='Nice';  
    num   pilote.pl#%TYPE;  
    nom   pilote.plnom%TYPE;  
    salaire   pilote.sal%TYPE;
```

```
BEGIN
```

```
    OPEN pl_nice;
```

```
    LOOP
```

```
        FETCH pl_nice INTO num, nom,salaire;
```

```
        ...
```

```
        EXIT WHEN sal > 10 000;
```

```
    END LOOP;
```

```
    CLOSE pl_nice;
```

```
END ;
```



## EXAMPLE: TYPE CURSOR

```
SET SERVEROUTPUT ON;  
DECLARE  
CURSOR CURSEUR1 IS SELECT * FROM etudiants;  
ligne CURSEUR1%rowtype;  
BEGIN  
OPEN CURSEUR1;  
LOOP  
FETCH CURSEUR1 INTO ligne;  
DBMS_OUTPUT.PUT_LINE('le nom est ' || ligne.nom || 'le  
prenom est ' ||  
LIGNE.prenom);  
EXIT WHEN curseur1%NOTFOUND;  
END LOOP;  
CLOSE CURSEUR1;  
END;
```

### 5.3. Les attributs d'un curseur

Pour tout curseur (implicite ou explicite) il existe des indicateurs sur leur état.

%FOUND

dernière ligne traitée

%NOTFOUND

%ISOPEN

ouverture d'un curseur

%ROWCOUNT nombre de lignes déjà traitées

IAM  
ouaga



*Label de réussite*

### 5.3. Les attributs d'un curseur

Pour tout curseur (implicite ou explicite) il existe des indicateurs sur leur état.

%FOUND

% found: contraire de %notfound:

Exemple:

```
CREATE OR REPLACE PROCEDURE test1insertion (NEMP IN VARCHAR2, PEMP IN VARCHAR2)AS
CURSOR CURSEUR1 IS SELECT * FROM TEST1;
NOMEMP VARCHAR2(30);
PRN VARCHAR2(30);
BEGIN
OPEN CURSEUR1;
LOOP
FETCH CURSEUR1 INTO NOMEMP,PRN;
IF CURSEUR1%FOUND THEN
INSERT INTO TEST1 VALUES(NEMP,PEMP);
COMMIT;
ELSE EXIT;
END IF;
END LOOP;
CLOSE CURSEUR1;
end test1insertion;
```

### 5.3. Les attributs d'un curseur

Pour tout curseur (implicite ou explicite) il existe des indicateurs sur leur état.

%ROWCOUNT

%rowcount: retourne le nombre d'enregistrements trouvés

Exemple:

```
LOOP
  FETCH emp_curseur INTO
    emp_nom, dept_num
    IF emp_cursor%rowcount>10 THEN
      EXIT;
    END IF;
END LOOP;
```

IAM  
ouaga



*Label de réussite*

### 5.3.1. %FOUND

#### curseur implicite : SQL%FOUND

TRUE

\* si INSERT, UPDATE, DELETE      traite au moins une ligne

\* si SELECT ... INTO ...              ramène une et une seule ligne

#### curseur explicite : nom\_curseur%FOUND

TRUE

\* si le dernier FETCH a ramené une ligne.

### 5.3.1. %FOUND (suite ...)

Exemple :

```
DECLARE
    CURSOR pl_nice IS
        SELECT pl#, plnom, sal
        FROM   pilote
        WHERE  adr='Nice';
    num   pilote.pl#%TYPE;
    nom   pilote.plnom%TYPE;
    salaire   pilote.sal%TYPE;

BEGIN
    OPEN pl_nice;
    FETCH pl_nice INTO num, nom,salaire;
    WHILE pl_nice%FOUND
    LOOP
        ...
        FETCH pl_nice INTO num,
            nom,salaire;

    END LOOP;
    CLOSE pl_nice;
END ;
```



## 5.3.2. %NOTFOUND

### curseur implicite : SQL%NOTFOUND

TRUE

\* si INSERT, UPDATE, DELETE      ne traite aucune ligne

\* si SELECT ... INTO ...      ne ramène pas de ligne

curseur explicite : nom\_curseur%NOTFOUND

TRUE

\* si le dernier FETCH n'a pas ramené de ligne.

### 5.3.3. %ISOPEN

#### curseur implicite : SQL%ISOPEN

toujours à FALSE car ORACLE referme les curseurs après utilisation.

curseur explicite : nom\_curseur%ISOPEN

TRUE si le curseur est ouvert.

Exemple 1 :

DECLARE

CURSOR pl\_nice IS

SELECT pl#, plnom, sal

FROM pilote

WHERE adr='Nice';

num pilote.pl#%TYPE;

nom pilote.plnom%TYPE;

salaire pilote.sal%TYPE;

BEGIN

IF NOT(pl\_nice%ISOPEN)

THEN

OPEN pl\_nice;

END IF;

...

END ;



### 5.3.3. %ISOPEN

curseur implicite : SQL%ISOPEN

%isopen: retourne vrai si le curseur est ouvert:

Exemple 2 :

```
IF emp_curseur%isopen THEN  
    FETCH ...  
ELSE  
    OPEN emp_curseur  
END IF
```

### 5.3.4. %ROWCOUNT

#### curseur implicite : SQL%ROWCOUNT

nombre de lignes traitées par INSERT, UPDATE, DELETE

- 0 : SELECT ... INTO : ne ramène aucune ligne
- 1 : SELECT ... INTO : ramène 1 ligne
- 2 : SELECT ... INTO : ramène plus d'une ligne

curseur explicite : nom\_curseur%ROWCOUNT

traduit la nième ligne ramenée par le FETCH

## 5.4. Simplification d'écriture

### 5.4.1. Déclaration de variables

Au lieu de déclarer autant de variables que d'attributs ramenés par le SELECT du curseur, on peut utiliser une structure.

Syntaxe :

```
DECLARE  
    CURSOR nom_curseur IS ordre_select;  
    nom_structure nom_curseur%ROWTYPE;
```

Pour renseigner la structure :

```
FETCH nom_curseur INTO nom_structure;
```

Pour accéder aux éléments de la structure :

```
nom_structure.nom_colonne
```

## 5.4. Simplification d'écriture

### 5.4.1. Déclaration de variables

Au lieu de déclarer autant de variables que d'attributs ramenés par le SELECT du curseur, on peut utiliser une structure.

Syntaxe :

```
DECLARE  
    CURSOR nom_curseur IS ordre_select;  
    nom_structure nom_curseur%ROWTYPE;
```

Pour renseigner la structure :

```
FETCH nom_curseur INTO nom_structure;
```

Pour accéder aux éléments de la structure :

```
nom_structure.nom_colonne
```

## 5.4.2. Traitement du curseur

Au lieu d'écrire :

```
DECLARE
    CURSOR nom_curseur IS SELECT ... ;
    nom_struct nom_curseur%ROWTYPE;
BEGIN
    OPEN nom_curseur;
    LOOP
        FETCH nom_curseur INTO nom_struct;
        EXIT WHEN nom_curseur%NOTFOUND;
        ...
    END LOOP;
    CLOSE nom_curseur;
END;
```

il suffit d'écrire :

```
DECLARE
    CURSOR nom_curseur IS SELECT ... ;
BEGIN
    FOR nom_struct IN nom_curseur LOOP
        ...
    END LOOP;
END;
```

## 5.4.2. Traitement du curseur (suite ...)

ou encore :

```
FOR nom_struct IN (SELECT ...)
```

```
  LOOP
```

```
    ...
```

```
  END LOOP;
```

**IAM**  
ouaga



*Label de réussite*

# 6. Gestion des erreurs en PL/SQL

La section EXCEPTION permet de gérer les erreurs survenues lors de l'exécution d'un bloc PL/SQL.

2 types d'erreurs :

- erreur ORACLE
- erreur utilisateur

**Syntaxe erreur utilisateur :**

```
DECLARE
    nom_erreur EXCEPTION;    on donne un nom à l'erreur
    ...
BEGIN
    IF ...
        THEN RAISE nom_erreur;    on déclenche l'erreur
        ...
        EXCEPTION
        WHEN nom_erreur THEN ... traitement de l'erreur
END;
```

Remarque : on sort du bloc après le traitement de l'erreur.

## 6. Gestion des erreurs en PL/SQL(suite)

Syntaxe erreur ORACLE non prédéfinie :

```
DECLARE
    nom_erreur EXCEPTION; on donne un nom à l'erreur
    PRAGMA EXCEPTION_INIT(nom_erreur,code_erreur)
        on associe le nom de l'erreur à un code erreur

    ...

BEGIN
    ...      l'erreur Oracle est détectée par le système
EXCEPTION
    WHEN nom_erreur THEN ... traitement de l'erreur
END;
```

Remarque :  
on sort du bloc après le traitement de l'erreur.



## 6. Gestion des erreurs en PL/SQL(suite)

### Syntaxe erreur ORACLE prédéfinie :

Certaines erreurs ORACLE ont déjà un nom. Il est donc inutile de les déclarer comme précédemment. On utilise leur nom dans la section EXCEPTION.

```
DECLARE
    ...
BEGIN
    ...      l'erreur Oracle est détectée par le système
EXCEPTION
    WHEN nom_erreur THEN ... traitement de l'erreur
END;
```

### Exemple d'erreurs prédéfinies :

- DUP\_VAL\_ON\_INDEX
- NO\_DATA\_FOUND
- ...
- OTHERS

## 6. Gestion des erreurs en PL/SQL(suite)

Les exeptions systèmes sont automatiquement soulevées lors de l'apparition de l'erreur ou de l'avertissement correspondant. Exemples d'exceptions système.

CURSOR\_ALREADY\_OPEN: tentative d'ouverture de curseur déjà ouvert

INVALID\_CURSOR: par exemple fetch sur un curseur déjà fermé

NO\_DATA\_FOUND: aucun tuple retourné (select into ou fetch)

TOO\_MANY\_ROWS: select into retourne plus d'un tuple ...

ZERO\_DIVIDE: tentative de division par zéro

Exemple d'usage

when NO DATA FOUND then rollback;

Les exception utilisateurs sont soulevées par raise

raise <nom d'exception>

IAM  
ouaga



*Label de réussite*

## 6. Gestion des erreurs en PL/SQL(suite)

Complément

- SQLCODE renvoie le code de l'erreur courante (numérique)
- SQLERRM[(code\_erreur)] renvoie le libellé de l'erreur courante ou le libellé de l'erreur dont le numéro est passé en paramètre.

Une erreur ou avertissement PL+SQL peut survenir en cours d'exécution et soulever une exception.

Une exception peut être prédéfinie par le système ou déclarée par l'utilisateur. Le traitement d'une exception se fait par la règle when

when <nom d'exception> then <sequence d'instructions>;

où la séquence d'instructions est exécuté quand l'exception donnée est soulevée.

## Traitement d'Exceptions (suite)

Les exceptions systèmes sont automatiquement soulevées lors de l'apparition de l'erreur ou de l'avertissement correspondant. Exemples d'exceptions système.

CURSOR\_ALREADY\_OPEN: tentative d'ouverture de curseur déjà ouvert

INVALID\_CURSOR: par exemple fetch sur un curseur déjà fermé

NO\_DATA\_FOUND: aucun tuple retourné (select into ou fetch)

TOO\_MANY\_ROWS: select into retourne plus d'un tuple ...

ZERO\_DIVIDE: tentative de division par zéro

### Exemple d'usage

```
when NO DATA FOUND then rollback;
```

Les exception utilisateurs sont soulevées par raise

```
raise <nom d'exception>
```

**ORACLE DATABASE :**  
**ORACLE PL/SQL et l'outil SQL\*PLUS**

**Merci pour votre attention**

**Questions**

