



# Chapitre 6

## Programmer avec pl/sql

**Enseignante: Nour H. BEN SLIMEN ATTAWI**

# Pourquoi un langage de programmation?

- ▶ SQL n'est pas un langage de programmation.
- ▶ SQL est un langage pensé pour permettre l'interrogation d'une base de manière déclarative, et non procédurale.
- ▶ Pas de **variable**, pas de **boucle**, pas de **test** .
- ▶ Pour écrire des applications, on utilise SQL associé à un langage de programmation

# PL/SQL

- ▶ PL/SQL est le langage procédural d'Oracle. Il est une extension du SQL qui est un langage ensembliste.
- ▶ PL/SQL est un langage structuré en blocs, constitués d'un ensemble d'instructions.
- ▶ Un bloc PL/SQL peut être "externe", on dit alors qu'il est anonyme, ou alors stocké dans la base de données sous forme de procédure, fonction ou trigger.
- ▶ Les instructions de manipulation des données, de contrôle des transactions, les fonctions SQL peuvent être utilisées avec la même syntaxe.
- ▶ Les instructions sont regroupées dans une unité appelée bloc qui ne génère qu'un accès à la base.
- ▶ Les blocs ou procédures PL/SQL sont compilés et exécutés par le moteur PL/SQL.
- ▶ En résumé, PL/SQL permet de construire des applications.

# Syntaxe d'un bloc PL/SQL

- ▶ Chaque bloc PL/SQL peut être constitué de 3 sections :
  - Une section facultative de déclaration et initialisation de types, variables et constantes
  - Une section obligatoire contenant les instructions d'exécution
  - Une section facultative de gestion des erreurs

## **DECLARE**

déclaration de variables, constantes et exceptions

## **BEGIN**

Les instructions à exécuter (commandes exécutables, instructions SQL et PL/SQL. Possibilité de blocs fils (imbrication de blocs))

## **EXCEPTION**

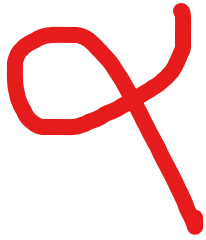
Traitement des exceptions (gestion des erreurs)

**END;**

# Exemple de bloc anonyme PL/SQL

L'exemple suivant montre un simple bloc anonyme PL/SQL avec une section exécutable.

## Exemple:



```
BEGIN  
    DBMS_OUTPUT.put_line ('Hello World!');  
END;
```

- ▶ La section exécutable appelle la procédure DBMS\_OUTPUT.PUT\_LINE pour afficher le message "Hello World" sur l'écran.

# LES VARIABLES

- 1 Types de variables**
- 2 Déclaration de variables**
- 3 Initialisation des variables**
- 4 Visibilité des variables**

# TYPE DE VARIABLES

- ▶ Les types habituels correspondants aux types Oracle :

✗ integer, number, varchar,...

- ▶ Types composites adaptés à la récupération des colonnes  
et lignes des tables SQL : %TYPE, %ROWTYPE, RECORD,  
TABLE

# TYPE DE VARIABLES

## DECLARE

```
v_remise CONSTANT real := 0.10;  
v_hiredate date;  
g_deptno number(2) NOT NULL := 10;  
v_integer integer; – max 38chiffres  
v_number1 number; – max 125 chiffres  
v_number2 number(38,3);  
v_bool boolean; – valeurs possibles TRUE, FALSE, NULL et NOT NULL  
v_char char(5); – Max 32767 caractères  
v_varchar2 varchar2(20); – Max 32767 caractères  
v_long LONG; – Max 2147483647 caractères  
v_date date; – Les dates peuvent aller de -4712 avant AVJC à 9999 APJC  
v_last_name employees.last_name%type;  
v_employee employees%rowtype  
v_n1 number(5,3);  
v_n2 v_n1%type;
```



# Type des variables

## Type RECORD

- ▶ Un enregistrement, ou un RECORD en PL/SQL, permet de regrouper dans un même type un ensemble d'informations caractéristiques d'un objet déterminé.
- ▶ Lors de la déclaration d'un enregistrement, il faut indiquer son nom et définir ses champs. Pour définir un champ, il faut indiquer son nom ainsi que le type d'information qu'il va contenir. Par exemple, supposons que vous disposiez de plusieurs renseignements au sujet d'un livre : son titre, son auteur, sa date de parution...
- ▶ Un type enregistrement défini dans un bloc PL/SQL est un type local. Il est disponible seulement dans le bloc où il est déclaré.
- ▶ Les enregistrements PL/SQL sont des types composites définis par l'utilisateur, Pour les utiliser il faut:
  - ▶ Déclarer l'enregistrement dans la section déclarative d'un bloc PL/SQL en utilisant le mot clé **TYPE**.
  - ▶ Déclarer et éventuellement initialiser les composants internes de ce type d'enregistrement
- ▶ Syntaxe:

```
TYPE record_name IS RECORD  
(field_1 [, field_2, ... field_N]... );  
  
new_record  record_name;
```

# DÉCLARATION DE VARIABLES

- Les variables locales se définissent dans la partie DECLARE d'un bloc PL/SQL.

VARIABLES PL/SQL de type oracle

## Exemple:

```
DECLARE
    nom          CHAR(15);
    numero       NUMBER;
    date_jour    DATE;
    salaire      NUMBER(7,2);

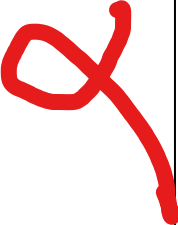
BEGIN
    . . .
END;
```

**N.B.:**  
~~Déclarations multiples interdites (i,j : number;)~~

# DÉCLARATION DE VARIABLES

- ▶ VARIABLES PL/SQL de type booléen

Exemple:



```
DECLARE
    reponse BOOLEAN :=true;

BEGIN
    . . .
END;
```

# DÉCLARATION DE VARIABLES

- ▶ VARIABLES faisant référence au dictionnaire de données :
  - ▶ Variables reprenant le même type qu'une colonne dans la table

## Syntaxe:



```
DECLARE
    nom_variable table.colonne%TYPE ;

BEGIN
    . . .

END ;
```

## Exemple:

```
DECLARE
    nameCl customers.name%TYPE ;

BEGIN
    . . .

END ;
```

CUSTOMERS
* CUSTOMER_ID
NAME
ADDRESS
WEBSITE
CREDIT_LIMIT

# DÉCLARATION DE VARIABLES

- ▶ VARIABLES faisant référence au dictionnaire de données :
  - ▶ Variables reprenant la même structure qu'une ligne d'une table

## Syntaxe:

```
DECLARE
nom_variable  table%ROWTYPE ;

BEGIN
    . . .
END;
```

## Exemple:

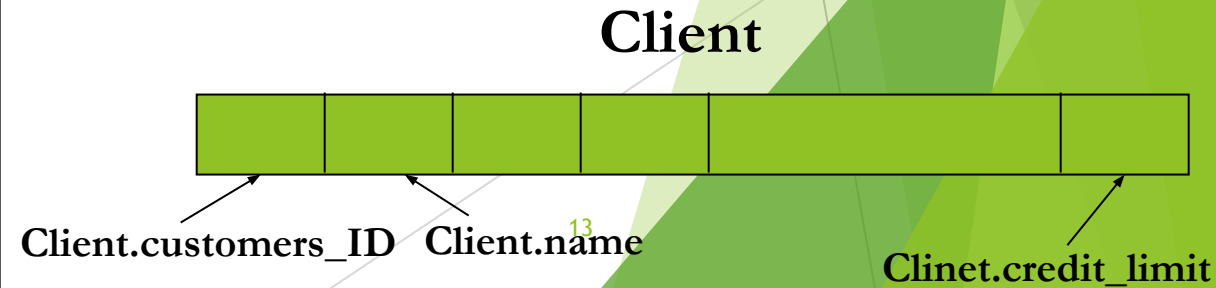
```
DECLARE
    Client customers%ROWTYPE ;

BEGIN
    . . .
END;
```

CUSTOMERS
* CUSTOMER_ID
NAME
ADDRESS
WEBSITE
CREDIT_LIMIT



Chaque variable de la structure Client  
a le même nom et le même type  
que  
la colonne associée




# DÉCLARATION DE VARIABLES

- ▶ VARIABLES faisant référence au dictionnaire de données :
  - ▶ Variables de même type qu'une variable précédemment définie :

## Syntaxe:

```
DECLARE  
nom_variable2 nom_variable1%TYPE;  
  
BEGIN  
    . . .  
END;
```

## Exemple:



```
DECLARE  
    commi  NUMBER(7,2) ;  
    salaire commi%TYPE ;  
  
BEGIN  
    . . .  
END;
```

# INITIALISATION DES VARIABLES

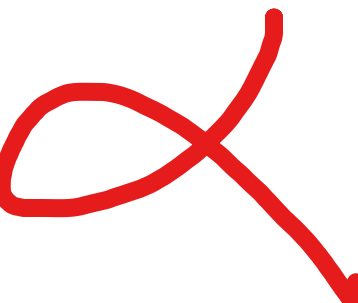
L'initialisation d'une variable peut se faire par :

- ▶ L'opérateur **:=** dans les sections **DECLARE**, **BEGIN** et **EXCEPTION**.
- ▶ L'ordre **SELECT ... INTO ....** dans la section **BEGIN**
- ▶ Le traitement d'un **curseur** dans la section **BEGIN**

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

# INITIALISATION DES VARIABLES


## ► L'opérateur :=



```
DECLARE
    nom      CHAR(10)      := 'Ben Amor' ;
    salaire  NUMBER(7,2)   := 1500 ;
    reponse   BOOLEAN      := TRUE ;
BEGIN
    ...
END;
```



Figer l'affectation de valeur à une variable avec la clause **CONSTANT**



```
DECLARE
    pi CONSTANT  NUMBER(7,2)   := 3.14 ;
BEGIN
    ...
END ;
```



# INITIALISATION DES VARIABLES

- Interdire les valeurs non renseignées avec la clause NOT NULL

α

```
DECLARE
    i  NUMBER  NOT NULL  :=  1000 ;
BEGIN
    ...
END   ;
;
```

# INITIALISATION DES VARIABLES

## L'ordre SELECT

### Syntaxe :



```
SELECT col1, col2 INTO var1, var2  
FROM table [WHERE condition ] ;
```

- ▶ La clause **INTO** est OBLIGATOIRE.
- ▶ Le SELECT doit obligatoirement ramener une seule ligne.
- ▶ Pour traiter un ordre SELECT qui permet de ramener plusieurs lignes, on utilise un **curseur**.

# INITIALISATION DES VARIABLES

## ► L'ordre SELECT


Syntaxe:

```
SELECT  
    select_list  
INTO  
    variable_list  
FROM  
    table_name  
WHERE  
    condition;
```

# INITIALISATION DES VARIABLES

## ► L'ordre SELECT

### Exemple:



```
DECLARE
    nom_emp CHAR(15) ;
    salaire      emp.sal%TYPE ;
    commision    emp.comm%TYPE ;
    nom_depart   CHAR(15) ;

BEGIN
    SELECT ename, sal, comm, dname
    INTO   nom_emp, salaire, commision, nom_depart
    FROM   emp, dept
    WHERE  ename = 'Hammami' AND
emp.deptno=dept.deptno ;
    ... .
END ;
```

# VISIBILITÉ DES VARIABLES

```
DECLARE
```

```
  compte  NUMBER(5) ;
```

```
  credit_max  NUMBER(9,2) ;
```

```
BEGIN
```

```
  .... ;
```

```
  DECLARE
```

```
    compte CHAR(20);
```

```
    balance1  NUMBER(9,2) ;
```

```
  BEGIN
```

```
    ....
```

```
  END;
```

```
  ....;
```

```
  DECLARE
```

```
    balance2  NUMBER(9,2) ;
```

```
  BEGIN
```

```
    ....
```

```
  END;
```

```
  ....
```

```
END ;
```

compte (NUMBER)  
credit\_max

compte (CHAR)  
balance1  
credit\_max

compte (NUMBER)  
balance2  
credit\_max

compte (NUMBER)  
credit\_max

# TRAITEMENTS CONDITIONNELS

**1. Définition et syntaxe**

**2. Exemple**

# TRAITEMENTS CONDITIONNELS

## 1. Définition et syntaxe



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



```
IF condition THEN  
    instructions 1;  
ELSE  
    instructions 2;  
END IF;
```



```
IF condition 1 THEN  
    instructions 1;  
ELSEIF condition 2 THEN  
    instructions 2;  
ELSEIF ...  
...  
ELSE  
    instructions N;  
END IF;
```

# TRAITEMENTS CONDITIONNELS

## 1. Exemple

```
DECLARE
    Emploi VARCHAR2(10) ;
    nom     VARCHAR2(15)      := 'Hammami' ;
    mes     VARCHAR2(30) ;
BEGIN
    SELECT job INTO emploi FROM emp WHERE ename=nom;
    IF emploi IS NULL THEN mes := nom||'n''a pas d''emploi';
    ELSIF emploi = 'Commercial'
    THEN UPDATE emp SET com = 1000 WHERE ename = nom;
        mes := nom || 'commision modifiée' ;
    ELSE UPDATE emp SET com = 0 WHERE ename = nom;
        mes := nom || 'pas de commision' ;
    END IF ;
    INSERT INTO resultat VALUES (mes) ;
    COMMIT;
END;
```

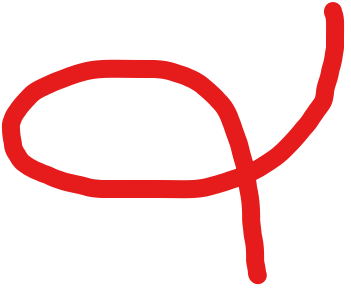


# TRAITEMENTS RÉPÉTITIFS

1. **La boucle de base (LOOP)**
2. **La boucle FOR**
3. **La boucle WHILE**

# TRAITEMENTS RÉPÉTITIFS

## La boucle de base (LOOP)



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

Exemple : Insérer les 10 premiers chiffres dans la table résultat.


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

# TRAITEMENTS RÉPÉTITIFS

## La boucle FOR

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

Exemple : Calcul de factorielle 9.



```
DECLARE
    fact    NUMBER    := 1 ;
BEGIN
    FOR i IN 1..9
        LOOP
            fact := fact * i ;
        END LOOP ;
        INSERT INTO resultat
        VALUES (fact, 'FACTORIELLE9');
    END ;
```

# TRAITEMENTS RÉPÉTITIFS

## La boucle WHILE

Condition est une condition d'expressions au moyen des opérateurs <, >, =, !=, AND, OR, LIKE, ...



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

Exemple : Afficher les 10 premiers chiffres .

```
DECLARE
    v1  NUMBER  :=  1 ;
BEGIN
    WHILE  v1<=10
    LOOP
        DBMS_OUTPUT.PUTLINE (v1) ;
        v1=v1+1;
    END LOOP ;
END ;
```

# LES CURSEURS

1. Définition
2. Les types de curseurs
3. Les étapes d'utilisation d'un curseur explicite
4. Les attributs d'un curseur
5. La boucle FOR pour un curseur

# LES CURSEURS

## Définition

- Pour traiter une commande SQL, PL/SQL ouvre une zone de contexte pour exécuter la commande et stocker les informations.
- Le curseur permet de nommer cette zone de contexte, d'accéder aux informations et éventuellement de contrôler le traitement.
- Cette zone de contexte est une mémoire de taille fixe, utilisée par le noyau pour analyser et interpréter tout ordre SQL.
- Les statuts d'exécution de l'ordre se trouvent dans le curseur.

# LES CURSEURS

## Les types des curseurs

***Le curseur explicite:*** Il est créé et géré par l'utilisateur pour traiter un ordre Select qui ramène plusieurs lignes. Le traitement du select se fera ligne par ligne.

***Le curseur implicite:*** Il est généré et géré par le noyau pour les autres commandes SQL.

# LES CURSEURS

## Les curseurs implicites

- Le curseur implicite est un curseur de session qui est déclaré et géré implicitement par PL/SQL.
- Le serveur Oracle ouvre un curseur implicite chaque fois qu'une instruction est exécutée. Cette instruction peut être :
  - SELECT, qui doit alors ramener exactement une seule ligne.
  - INSERT, DELETE ou UPDATE sans aucune contrainte sur le nombre de n-uplets affectés.



# LES CURSEURS

## Les curseurs explicites

- 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.

# LES CURSEURS

## Les curseurs explicites

- L'utilisation d'un curseur pour traiter un ordre **Select** ramenant plusieurs lignes, nécessite 4 étapes :

**1. Déclaration du curseur**

**2. Ouverture du curseur**

**3. Traitement des lignes**

**4. Fermeture du curseur.**

# LES CURSEURS

## Déclaration d'un curseur explicite

### Déclaration:

Tout curseur explicite utilisé dans un bloc PL/SQL doit être déclaré dans la section **DECLARE** du bloc en donnant :

- ▶ Son nom
- ▶ L'ordre select associé

### Syntaxe

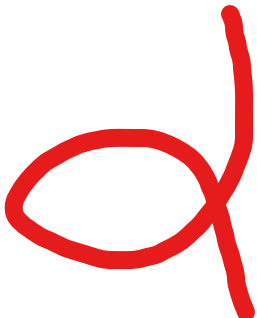


```
CURSOR nom_curseur IS ordre_select;
```

# LES CURSEURS

## Déclaration d'un curseur explicite

### Exemple :



```
DECLARE

    CURSOR dept_10 IS

        Select ename, Sal FROM emp

        WHERE deptno = 10

        ORDER BY sal;

BEGIN

    ... ;

    ... ;

END;
```

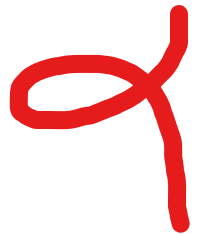
# LES CURSEURS

## Ouverture d'un curseur explicite

### Ouverture

- ▶ Après avoir déclaré le curseur, il faut l'ouvrir pour faire exécuter l'ordre **SELECT**.
- ▶ L'ouverture du curseur se fait dans la section **BEGIN** du bloc.

### Syntaxe:




```
OPEN nom_curseur;
```

# LES CURSEURS

## Ouverture d'un curseur explicite

Exemple:



```
DECLARE
    CURSOR dept_10 IS
    Select ename, Sal FROM emp
    WHERE deptno = 10
    ORDER BY sal;

BEGIN
    ... ;
    OPEN dept_10;
    ... ;
END;
```

# LES CURSEURS

## 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;
```



Le Fetch ramène une seule ligne à la fois.

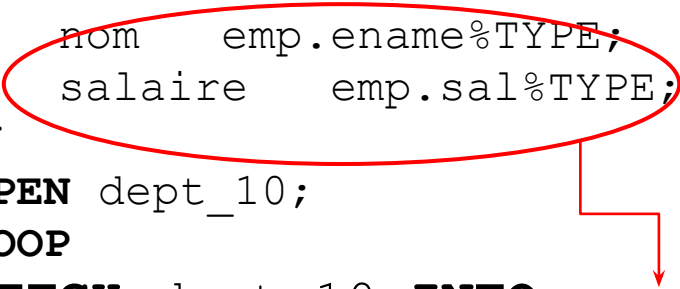
Pour traiter n lignes, il faut prévoir une boucle

# LES CURSEURS

## Traitement des lignes

### Exemple:

```
DECLARE
    CURSOR dept_10 IS
        Select ename,sal FROM emp
        WHERE deptno = 10
        ORDER BY sal;
        nom      emp.ename%TYPE;
        salaire   emp.sal%TYPE;
BEGIN
    OPEN dept_10;
    LOOP
        FETCH dept_10 INTO nom, salaire;
        IF salaire >2500 THEN
            INSERT INTO resultat VALUES (nom,salaire);
        END IF;
        EXIT WHEN salaire = 5000;
    END LOOP;
END;
```



Variables réceptrices qui  
correspondent aux attributs  
du curseur



# LES CURSEURS

## Fermeture du curseur

- ▶ Après le traitement des lignes, on ferme le curseur pour libérer la place mémoire

### Syntaxe

```
CLOSE nom_curseur;
```

# LES CURSEURS

## Fermeture du curseur

Exemple :

```
DECLARE
    CURSOR dept_10 IS Select ename,sal FROM emp
    WHERE deptno = 10
    ORDER BY sal;
    nom emp.ename %TYPE;
    salaire emp.sal %TYPE;

BEGIN
    OPEN dept_10;
    LOOP
        FETCH dept_10 INTO nom, salaire;
        IF salaire >2500 THEN INSERT INTO resultat VALUES
(nom,salaire);
        END IF;
        EXIT WHEN salaire = 5000;
    END LOOP;
    CLOSE dept_10;
END;
```

# LES CURSEURS

## Les attributs d'un curseur

- ▶ Tous les curseurs ont des attributs que l'utilisateur peut utiliser. Les attributs d'un curseur sont des indicateurs sur l'état d'un 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).
  - ▶ **%ROWCOUNT** Nombre de lignes traitées par le curseur.

# LES CURSEURS

## Les attributs d'un curseur

### Attribut %FOUND

- Type: booléen
- Syntaxe:
  - Curseur implicite : SQL%FOUND
  - Curseur explicite : nom\_curseur%FOUND

Si sa valeur est vrai donc le dernier FETCH a ramené une ligne

# LES CURSEURS

## Les attributs d'un curseur

### Attribut %FOUND

#### Exemple:

```
DECLARE
    CURSOR dept_10 IS Select ename, sal FROM emp
    WHERE deptno = 10 ORDER BY sal;
    nom emp.ename%TYPE;
    salaire emp.sal%TYPE;

BEGIN
    OPEN dept_10;
    FETCH dept_10 INTO nom, salaire;

    WHILE dept_10%FOUND
    LOOP
        IF salaire > 2500 THEN
            INSERT INTO resultat VALUES (nom, salaire);
        END IF;
        FETCH dept_10 INTO nom, salaire;
    END LOOP;
    CLOSE dept_10;
END;
```

# LES CURSEURS

## Les attributs d'un curseur

### Attribut %NOTFOUND

- Type: booléen
- Syntaxe:
  - Curseur implicite : SQL%NOTFOUND
  - Curseur explicite : nom\_curseur%NOTFOUND

Si sa valeur est vrai donc le dernier FETCH n'a pas ramené une ligne

# LES CURSEURS

## Les attributs d'un curseur

### Attribut %NOTFOUND

Exemple:

```
DECLARE
    CURSOR dept_10 IS Select ename,sal FROM emp
    WHERE deptno = 10 ORDER BY sal;
    nom emp.ename%TYPE;
    salaire emp.sal%TYPE;

BEGIN
    OPEN dept_10;
    LOOP
        FETCH dept_10 INTO nom, salaire;
        IF salaire >2500 THEN
            INSERT INTO resultat VALUES (nom,salaire);
        END IF;
        EXIT WHEN DEPT_10%NOTFOUND;
    END LOOP;
    CLOSE dept_10;
END;
```

# LES CURSEURS

## Les attributs d'un curseur

### Attribut %ISOPEN

- Type: booléen
- Syntaxe:
  - **Curseur implicite** : SQL%ISOPEN  
toujours à **FALSE** car Oracle referme les curseurs après utilisation
  - **Curseur explicite** : nom\_curseur%ISOPEN  
Si sa valeur est vrai donc le curseur est ouvert



# LES CURSEURS

## Les attributs d'un curseur

### Attribut %ISOPEN

Exemple:

```
DECLARE
    CURSOR dept_10 IS
        Select ename, sal FROM emp WHERE deptno = 10 ORDER
        BY sal;
        nom emp.ename%TYPE;
        salaire emp.sal%TYPE;
BEGIN
    IF NOT (dept_10%ISOPEN) THEN
        OPEN dept_10;
    END IF;
    LOOP
        FETCH dept_10 INTO nom, salaire;
        EXIT WHEN DEPT_10%NOTFOUND;
        IF salaire > 2500 THEN
            INSERT INTO resultat VALUES (nom,salaire);
        END IF;
    END LOOP;
    CLOSE dept_10;
END;
```

# LES CURSEURS

## Les attributs d'un curseur

### Attribut %ROWCOUNT

- Type: numérique
- Syntaxe:
  - **Curseur explicite** : nom\_curseur%ROWCOUNT.  
Traduit la n<sup>ième</sup> ligne ramenée par le **FETCH**

# LES CURSEURS

## Les attributs d'un curseur

### Attribut %ROWCOUNT

#### Exemple:

```
DECLARE
    CURSOR dept_10 IS
        SELECT ename, sal FROM emp WHERE deptno = 10 ORDER BY
            sal;
    nom emp.ename%TYPE;
    salaire emp.sal%TYPE;
BEGIN
    OPEN dept_10;
    LOOP
        FETCH dept_10 INTO nom, salaire;
        EXIT WHEN DEPT_10%NOTFOUND OR DEPT_10%ROWCOUNT>15 ;
        IF salaire > 2500 THEN INSERT INTO resultat VALUES
            (nom, salaire);
        END IF;
    END LOOP;
    CLOSE dept_10;
END;
```

# LES CURSEURS

## Les attributs d'un curseur

### Attribut %ROWTYPE

Cet attribut permet la déclaration implicite d'une structure dont les éléments sont d'un type identique aux colonnes ramenées par le curseur.

#### Syntaxe :

- Dans la partie déclarative du bloc.

```
CURSOR nomcurseur IS ordre_select;  
nomrecord nomcurseur%Rowtype;
```

- Les éléments de la structure sont identifiés par :

```
nomrecord.nomcolonne
```

- La structure est renseignée par le Fetch :

```
FETCH nomcurseur INTO nomrecord;
```

# LES CURSEURS

## Les attributs d'un curseur

### Attribut %ROWTYPE

#### Exemple:

```
DECLARE
    CURSOR c1 IS Select ename,sal FROM emp
    WHERE deptno = 10 ORDER BY sal;
    c1_rec c1%ROWTYPE;
BEGIN
    OPEN c1;
    LOOP
        FETCH c1 INTO c1_rec;
        EXIT WHEN c1%NOTFOUND
        IF c1_rec.sal > 2500 THEN      INSERT INTO resultat
            VALUES (c1_rec.ename, c1_rec.sal);
        END IF;
    END LOOP;
    CLOSE c1;
END;
```

# LES CURSEURS

## Boucle FOR pour un curseur

- Elle simplifie la programmation car elle évite d'utiliser explicitement les instructions OPEN, FETCH et CLOSE
- Elle déclare implicitement une variable de type « row » associée au curseur.

### Syntaxe:

```
DECLARE
  CURSOR nom_curseur IS ordre_select;
BEGIN
  FOR nom_rec IN nom_curseur LOOP
    /*-----Traitements-----
    -----*/
  END LOOP;
END;
```


# LES CURSEURS

## Boucle FOR pour un curseur

### Exemple :

```
DECLARE
    CURSOR dept_10 IS
        Select ename, sal FROM emp WHERE deptno = 10
        ORDER BY sal;
BEGIN
    FOR nom_rec IN dept_10;
    LOOP
        dbms_output.put_line ('Employé:' || nom_rec.ename
                               || 'et' || 'Salaire:' || nom_rec.sal);
    END LOOP;
END;
```

Variable de type  
dept\_10%ROWTYPE



# GESTION DES EXCEPTIONS

1. Introduction
2. Les exceptions internes
3. Les exceptions utilisateurs



# GESTION DES EXCEPTIONS

## Introduction

- ▶ Le mécanisme de gestion d'erreurs dans PL/SQL est appelé gestionnaire des exceptions.
- ▶ Il permet au développeur de planifier sa gestion et d'abandonner ou de continuer le traitement en présence d'une erreur.
- ▶ Il faut affecter un traitement approprié aux erreurs apparues dans un bloc PL/SQL.

# GESTION DES EXCEPTIONS

## Introduction

On distingue 2 types d'erreurs ou d'exceptions :

- ▶ Erreur interne : dans ce cas la main est rendue directement au système environnant.
- ▶ Anomalie déterminée par l'utilisateur.

La solution :

- ▶ Donner un nom à l'erreur (si elle n'est pas déjà prédéfinie),
- ▶ Définir les anomalies utilisateurs, leur associer un nom,
- ▶ Définir le traitement à effectuer.

# GESTION DES EXCEPTIONS

## Les exceptions internes

- ▶ Une erreur **interne** est produite quand un bloc PL/SQL viole une règle d'Oracle ou dépasse une limite dépendant du système d'exploitation.
- ▶ Les erreurs Oracle générées par le noyau sont numérotées, or le gestionnaire des exceptions de PL/SQL ne sait que gérer des erreurs nommées.
- ▶ Pour cela PL/SQL a redéfini quelques erreurs Oracle comme des exceptions.

# GESTION DES EXCEPTIONS


## Les exceptions définies par le système

Erreur	Nom d'exception	Soulevé quand
ORA-00001	DUP_VAL_ON_INDEX	Une valeur en double existe
ORA-01001	INVALID_CURSOR	Le curseur n'est pas valide
ORA-010121	NOT_LOGGED_ON	L'utilisateur n'est pas connecté
ORA-01017	LOGIN_DENIED	Une erreur système s'est produite
ORA-01017	LOGIN_DENIED	Une erreur système s'est produite
ORA-01017	LOGIN_DENIED	Une erreur système s'est produite
ORA-01403	NO_DATA_FOUND	La requête ne renvoie aucune donnée
ORA-01422	TOO_MANY_ROWS	Une requête sur une seule ligne renvoie plusieurs lignes
ORA-01476	ZERO_DIVIDE	Une tentative a été faite pour diviser un nombre par zéro
ORA-01722	INVALID_NUMBER	Le numéro est invalide
ORA-06504	ROWTYPE_MISMATCH	Une incompatibilité s'est produite dans le type de ligne
ORA-06511	CURSOR_ALREADY_OPEN	Le curseur est déjà ouvert
ORA-06532	COLLECTION_IS_NULL	Travailler avec la collection NULL
ORA-06531	SUBSCRIPT_OUTSIDE_LIMIT	Index de collection hors de portée
>ORA-06533	SUBSCRIPT_BEYOND_COUNT	Index de collection hors de compte

# GESTION DES EXCEPTIONS

## Les exceptions internes

### Exemple:



```
DECLARE
    Nom varchar2 (100);
    Salaire number;
    EmpID number := 8376;

BEGIN
    SELECT Ename, sal INTO Nom, Salaire FROM Emp WHERE EmpNo =
    EmpID;

    EXCEPTION

        WHEN NO_DATA_FOUND THEN
            dbms_output.putline('Aucun employé n''existe avec l''
            identifiant' || EmpID);

END ;
```

# GESTION DES EXCEPTIONS

## Les exceptions déterminées par l'utilisateur

- ▶ PL/SQL permet à l'utilisateur de définir **ses propres exceptions**.
- ▶ La gestion des anomalies utilisateur peut se faire dans un bloc PL/SQL en effectuant les opérations suivantes :

❶ **Nommer l'anomalie** (type exception) dans la partie DECLARE du bloc.

```
DECLARE
```

```
Nom_ano EXCEPTION;
```

❷ **Déterminer l'erreur** et passer la main au traitement approprié par la commande **RAISE**.

```
BEGIN
```

```
...
```

```
IF (condition_anomalie) THEN RAISE Nom_ano ;
```

❸ **Effectuer le traitement** défini dans la partie EXCEPTION du Bloc.

```
EXCEPTION
```

```
WHEN (Nom_ano) THEN (traitement);
```

# GESTION DES EXCEPTIONS

## Les exceptions internes

### Exemple :

**DECLARE**

a number :=5;

b number :=0;

res number;

*/\* Déclarer une exception \*/*

**erreur EXCEPTION;**

**BEGIN**

**IF** b = 0 **THEN**

**RAISE erreur;**

**END IF;**

**EXCEPTION**

**WHEN** erreur **THEN** ...;

**dbms\_output.put\_line**('La valeur de b doit être différente de zéro');

**END;**