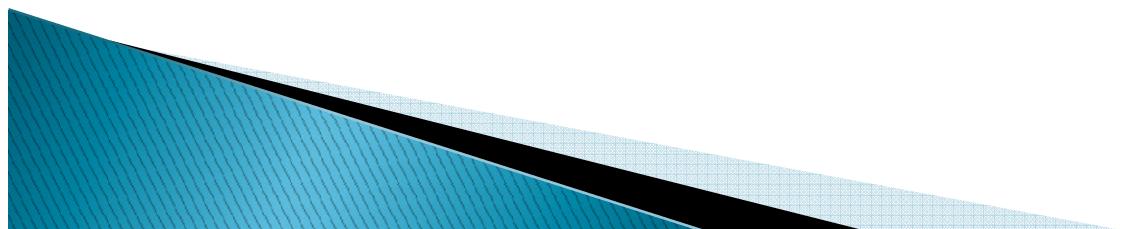
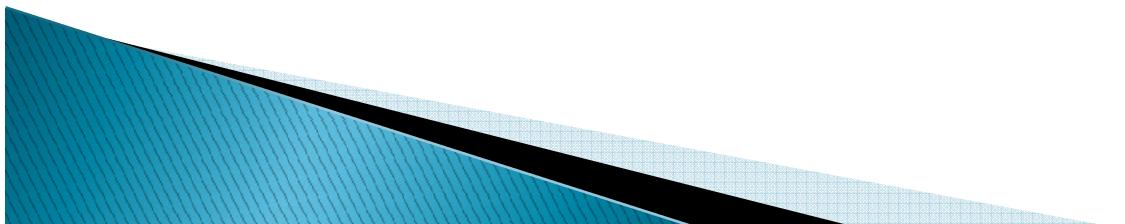


# Cours PL/SQL



# Introduction

- Langage propre à Oracle basé sur ADA
- Offre une extension procédurale à SQL
- PL/SQL permet d'utiliser un sous-ensemble du langage SQL des variables, des boucles, des alternatives, des gestions d'erreurs.
- PL/SQL permet également de manipuler ligne par ligne les données renvoyées par une requête SQL.



# Bloc PL/SQL

Un programme PL/SQL est composé d'un ou plusieurs blocs.

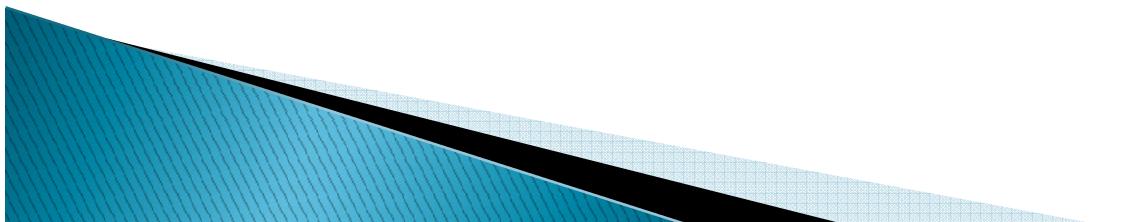
Chaque bloc comporte 4 sections.

## *1 Section en-tête (facultative)*

Permet de nommer le bloc (bloc anonyme ou bloc nommé : procédure, fonction, paquage,...)

PROCEDURE nom[(liste de paramètres)] IS|AS ...

FUNCTION nom[(liste de paramètres)] RETURN nomType  
IS|AS ...

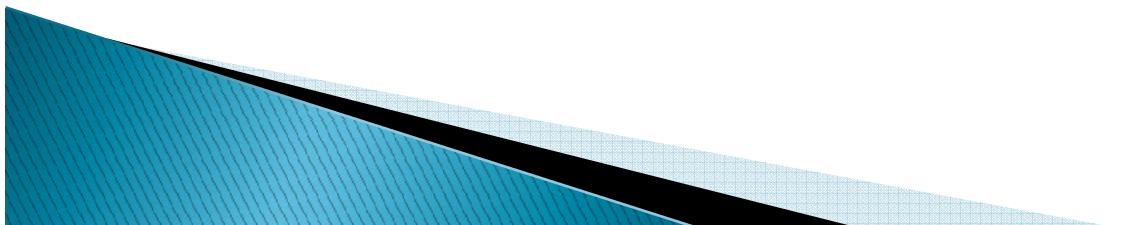


# Bloc PL/SQL

## *2 Section déclaration (facultative)*

Introduite par le mot clé « DECLARE ». On fait ici les déclarations des constantes, des variables, des exceptions, des curseurs.

**DECLARE ...**



# Bloc PL/SQL

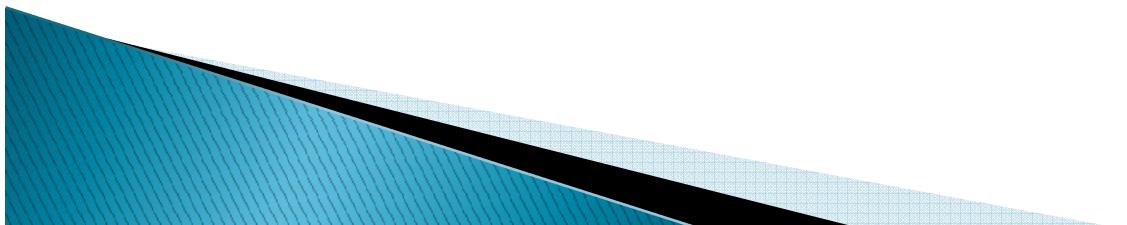
## *3 Section corps de programme (obligatoire)*

Commence par « BEGIN » et se termine par « END ».

BEGIN

...

END



# Bloc PL/SQL

## *4 Section des exceptions (facultative)*

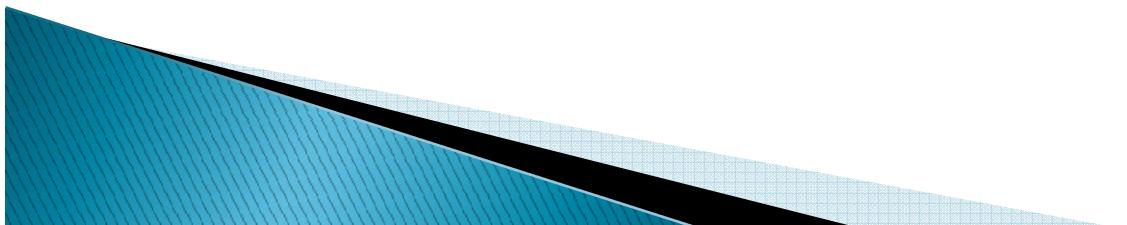
Dans la section corps du programme, elle est introduite par le mot clé « EXCEPTION ».

BEGIN

Traitement

**EXCEPTION ...**

END



# *Type des données*

PL/SQL offre 2 grandes familles de type de données.

**Types scalaires (lignes ne retournant qu'une seule valeur)**

**Types de base**

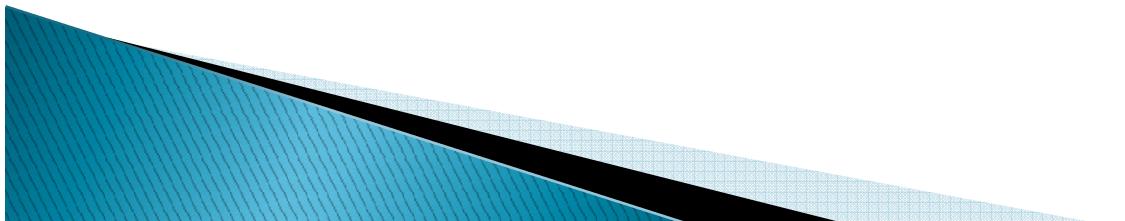
INTEGER, NUMBER, FLOAT, REAL, ...,  
CHAR, VARCHAR2, DATE, BOOLEAN

*Nom\_variable nom\_type*

**Variables basées**

**%TYPE** sert à baser la variable sur une autre structure de données : une variable PL/SQL, un attribut d'une relation, un curseur.

*Nom\_nouvelle\_variable nom\_variable %TYPE*



# *Type des données*

## **Variables basées**

Ex :

*Base de données :*

*Employé (nom VARCHAR2(30), Salaire  
NUMBER(8,2))*

*Bloc PL/SQL :*

*DECLARE*

*v\_nom    Employe.nom    %TYPE  
v\_new\_nom    v\_nom    %TYPE*

*v\_nom est en fait basé sur le type de l'attribut nom  
de la table Employé, i.e. VARCHAR2(30) et  
v\_new\_nom est basé sur le type de v\_nom.*



# Type des données : Types composés

## Enregistrement

1. Par numérotation des champs qui composent l'enregistrement

```
TYPE nom_type IS RECORD  
(nom_champ1 type1,  
 ...  
 nom_champN typeN)
```

2. Enregistrement basé sur une relation

Utilisation de %ROWTYPE.

Les enregistrements sont semblables, en terme de lignes, à la structure d'une relation.

```
v_employe employe      %ROWTYPE
```

Accès au champ : *nom\_variable.nom\_champi*

3. Enregistrement basé sur un curseur

Structure calquée sur la liste du SELECT ramené par le curseur.

```
nom_variable      nom curseur      %ROWTYPE
```

# *Type des données :* Types composés

- ▶ Tableau
- ▶ Une seule dimension indexé par BINARY INTEGER
- ▶ Eléments de même type

```
TYPE nom_type IS TABLE OF type_des_elements  
INDEX BY BINARY INTEGER  
nom_variable nom_type  
nom_variable(2)
```

## Constantes

```
nom_constante CONSTANT type := valeur ;
```

# Instructions

## Instructions SQL

INSERT, UPDATE, DELETE

SELECT FROM WHERE

Gestion des transactions : COMMIT, ROLLBACK

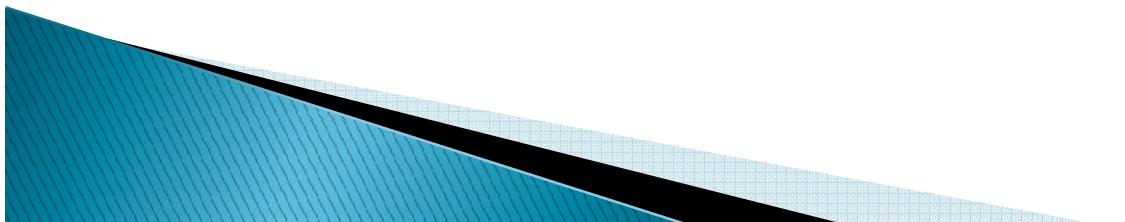
## Instructions d'affectation

## Instructions de contrôle de données

## Instructions de gestion des curseurs

## Instructions de gestion des exceptions

Remarque : Chaque instruction doit se terminer par « ; »



# *Ordres d'affectation*

1. `:=`

*nom\_variable := valeur ;*

2. Valeur de résultat d'une requête

*SELECT ...*

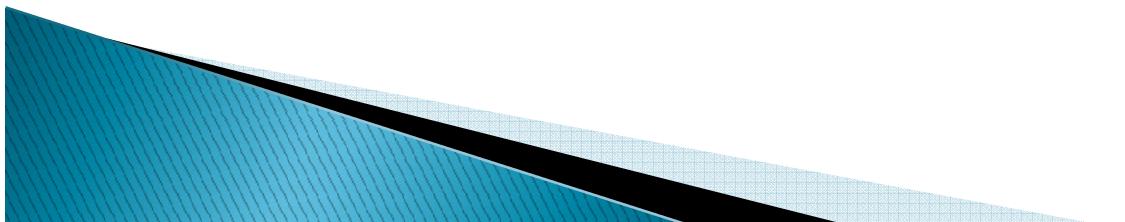
*INTO liste de variables*

*FROM*

Attention : Utilisable que si on est sûr que le SELECT ramènera une et une seule ligne.

3. Fetch

L'ordre Fetch qui permet de récupérer 1 ligne d'un curseur.



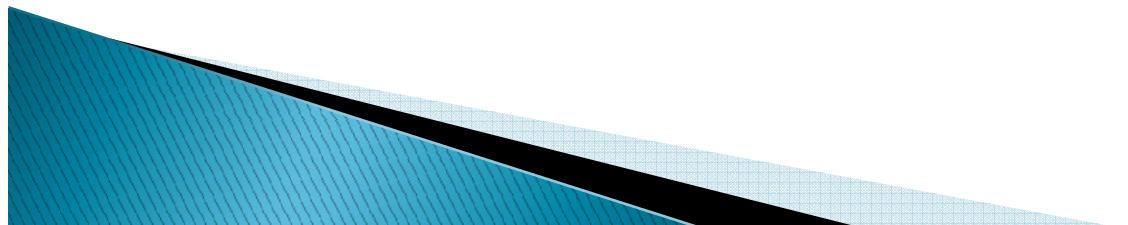
# *Structures de contrôle*

**Alternative**

```
IF condition THEN  
    instruction1;  
ENDIF;
```

```
IF condition THEN  
    instruction1;  
ELSE  
    instruction2 ;  
ENDIF;
```

```
IF condition THEN  
    instruction1;  
    ELSIF condition2 then instruction2;  
    ELSIF condition3...  
  
ELSE  
    instructionN ;  
ENDIF;
```



# *Structures de contrôle*

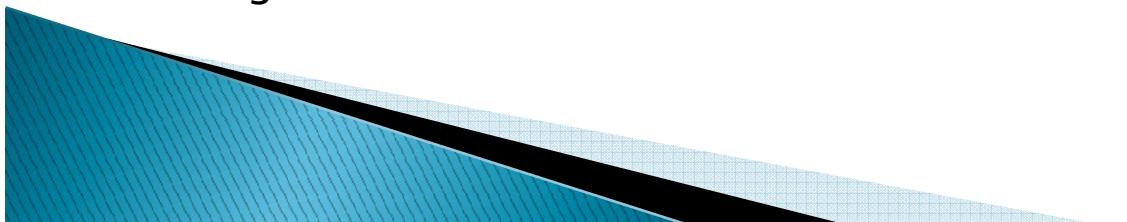
## Répétition

- ▶ *LOOP*  
    *instruction;*  
    *EXIT [WHEN condition] ; (ou IF condition THEN EXIT;)*  
    *instruction;*  
    *END LOOP;*
  
- ▶ *WHILE condition LOOP*  
    *instruction;*  
    *END LOOP ;*
  
- ▶ *FOR (variable\_indice) IN [REVERSE] Min .. Max LOOP*  
    *instruction;*  
    *END LOOP;*

Variable<sub>\_indice</sub> : variable locale à la boucle, elle n'a pas besoin d'être déclarée

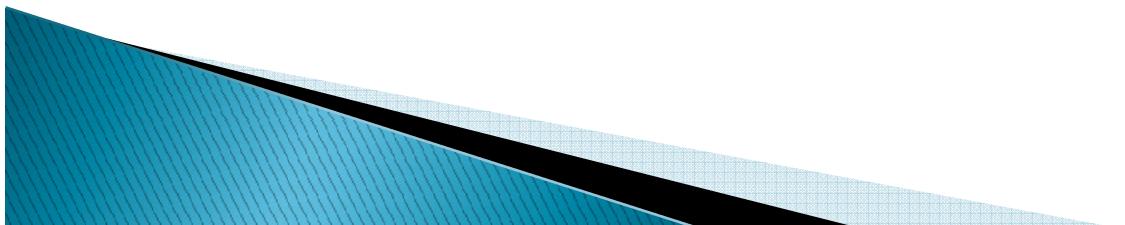
REVERSE : va de Max à Min

Min, Max : constantes ou des variables déclarées et affectées, le pas est obligatoirement de 1.



# Extraire des données

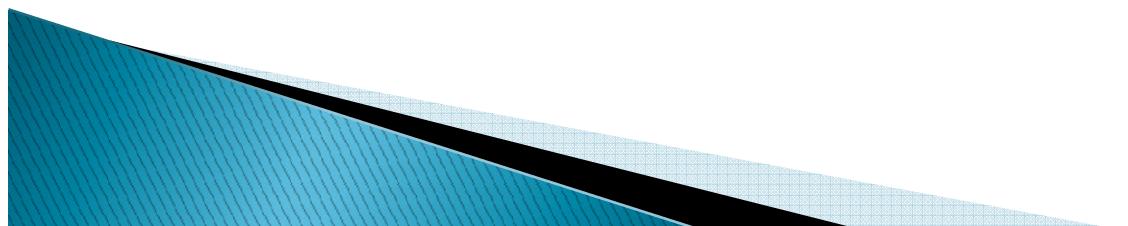
- ▶
- **select expr1, expr2,... into var1, var2,...**
- ▶ met des valeurs de la BD dans une ou plusieurs variables
- Le select ne doit renvoyer qu'une seule ligne
- Avec Oracle il n'est pas possible d'inclure un select sans « into » dans une procédure ;
- ▶



## Extraire des données – erreurs

Si le select renvoie plus d'une ligne, une exception «**TOO\_MANY\_ROWS** » (ORA-01422) est levée

Si le select ne renvoie aucune ligne, une exception «**NO\_DATA\_FOUND** » (ORA-01403) est levée



# Exemple

*DECLARE*

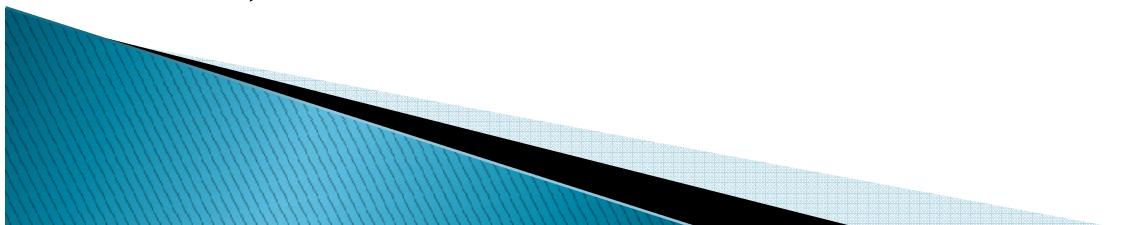
```
v_nom emp.nome %TYPE;  
v_emp emp %ROWTYPE;
```

*BEGIN*

```
select nome into v_nom  
from employe  
where budget = 500;
```

```
select * into v_emp  
from emp  
where budget = 500;
```

*END;*



# Les curseurs

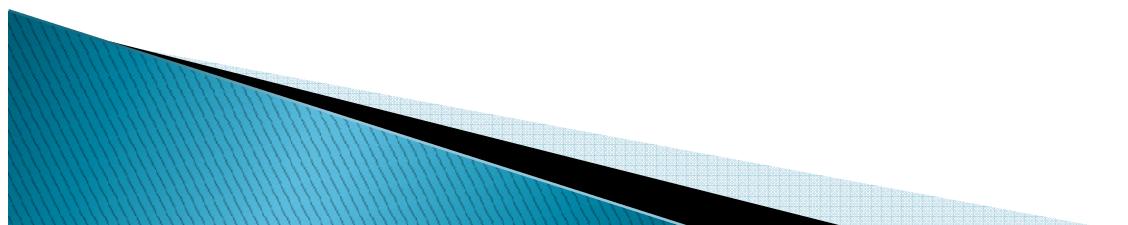
Un curseur permet de récupérer et de traiter ligne par ligne un ensemble de lignes de la BD.

En PL/SQL, il existe 2 types de curseurs :

**les curseurs implicites** : représente la zone mémoire utilisée pour parser et exécuter la requête

Dès l'instant où l'on exécute un ordre SQL, il y a création d'un curseur. Un curseur est une zone de travail (« Context area ») qui contient toutes les infos relatives à l'ordre SQL (texte source de la requête, forme traduite, plan d'exécution, résultat...)

**les curseurs explicites** (gérés par le programme PL/SQL, ils permettent de traiter une à une les lignes ramenées avec un select),



# *Curseurs explicites*

Déclaration d'un curseur explicite

*DECLARE*

*CURSOR nom curseur IS requete\_SQL ;*

*CURSOR e IS SELECT \* FROM Employe;*

-- Un curseur peut être défini avec des paramètres

**CURSOR**

*nom curseur(liste\_param=val\_defaut) IS  
requete\_SQL ;*

*Ex : CURSOR e1(Sal number) IS SELECT \*  
FROM Employe WHERE Salaire>\*Sal;*

# *Curseurs explicites*

## Ouverture de curseur

Alloue l'espace mémoire suffisant et permet de positionner d'éventuels verrous

```
OPEN nom curseur ;  
OPEN nom curseur(liste de paramètres réels) ;
```

La requête est exécutée à ce moment là.

## Fermeture du curseur

```
CLOSE nom curseur ;
```

## Traitement des lignes

Distribution une à une des lignes par l'ordre FETCH placé à l'intérieur d'une boucle.

```
FETCH nom curseur INTO variable ;
```



# les statuts d'un curseur

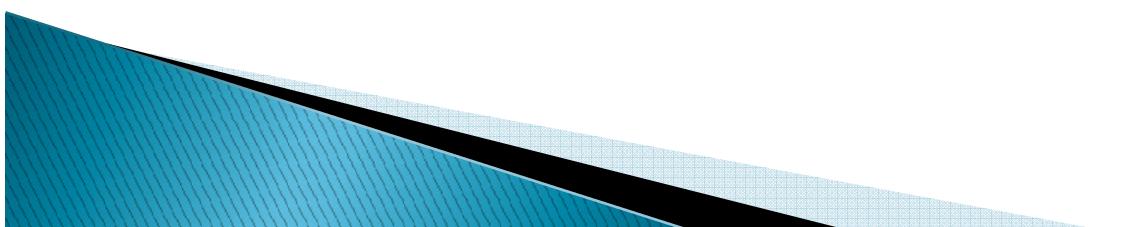
%ISOPEN : renvoie VRAI si le curseur est ouvert

%FOUND : renvoie VRAI si le dernier FETCH a renvoyé  
1

ligne

%NOTFOUND

%ROWCOUNT : renvoie le nombre de lignes traitées  
par le curseur

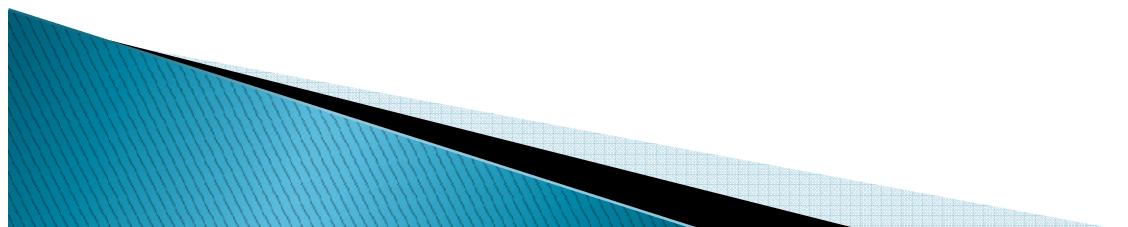


# Les curseurs implicites

Les curseurs implicites sont tous nommés SQL

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

...



# Exemple

```
DECLARE
    CURSOR e IS SELECT * FROM Employe;
    v_nom Employ.nom %TYPE ;
    v_salaire Employ.salaire %TYPE ;
BEGIN
    OPEN e;
    LOOP
        FETCH e INTO v_nom, v_salaire;
        EXIT WHEN e%NOTFOUND;
        Traitement
    END LOOP;
    CLOSE e;
END;
```

Remarque : Quand on veut traiter une à une, sans exception, les lignes retournées il faut utiliser une boucle “FOR”.

**FOR indice IN nom curseur**

indice n'a pas besoin d'être déclaré.

OPEN et CLOSE sont exécutés de manière implicite

# Exemple

*DECLARE*

*CURSOR e IS SELECT \* FROM Employe;*

*v\_nom Employ.nom %TYPE*

*v\_salaire Employe.salaire %TYPE*

*BEGIN*

*FOR i IN e*

*LOOP*

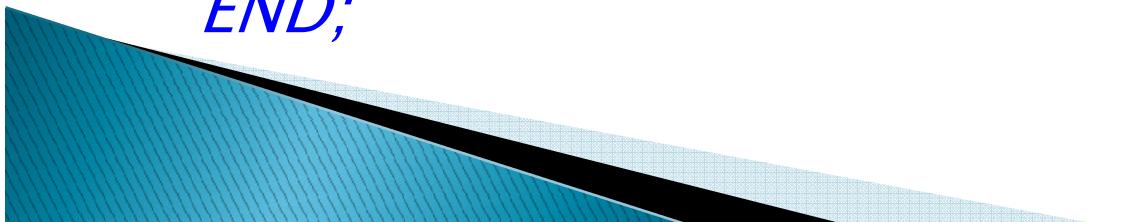
*v\_nom := i.nom;*

*v\_salaire := i.salaire;*

*Traitement*

*END LOOP;*

*END;*



# Modification de données

On peut effectuer une modification (UPDATE ou DELETE) sur la ligne que renvoie le curseur grâce à la commande « CURRENT OF ».

**UPDATE | DELETE ... WHERE CURRENT OF nom 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 [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



# Modification de données

Ex :

*DECLARE*

*CURSOR c IS SELECT \* FROM Employe [FOR UPDATE OF Commission];*

*v\_nom Employ.nom %TYPE*

*v\_salaire Employe.salaire %TYPE*

*v\_commission Employe.commission %TYPE*

*BEGIN*

*OPEN c;*

*LOOP*

*FETCH c INTO v\_nom, v\_salaire, v\_commission;*

*EXIT WHEN c %NOTFOUND;*

*IF v\_commission IS NULL THEN*

*UPDATE Employe SET commission:=v\_salaire \* 0.1*

*WHERE CURRENT OF c ;*

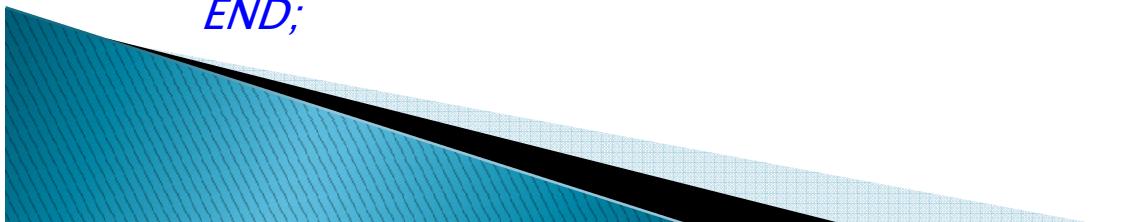
*END IF ;*

*END LOOP;*

*CLOSE e;*

*COMMIT;*

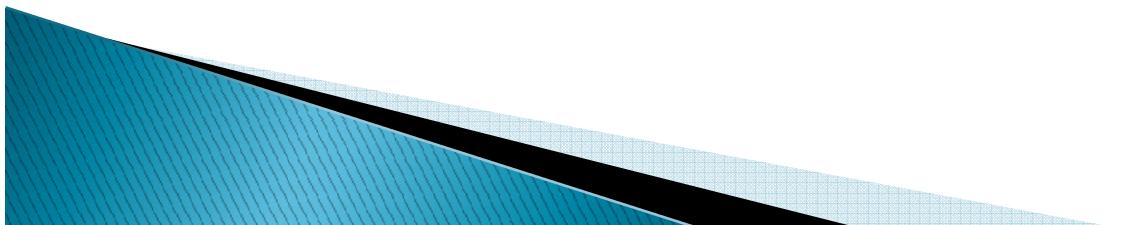
*END;*



# Exceptions

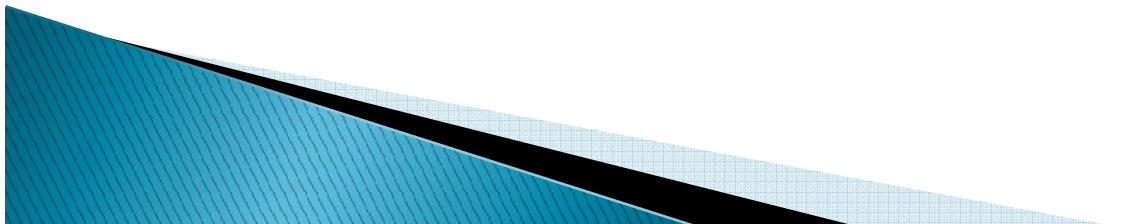
- ▶ Une exception est une erreur qui survient durant une exécution
- ▶ 2 types d'exception :
  - prédéfinie par Oracle
  - définie par le programmeur

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



# Exceptions prédéfinies

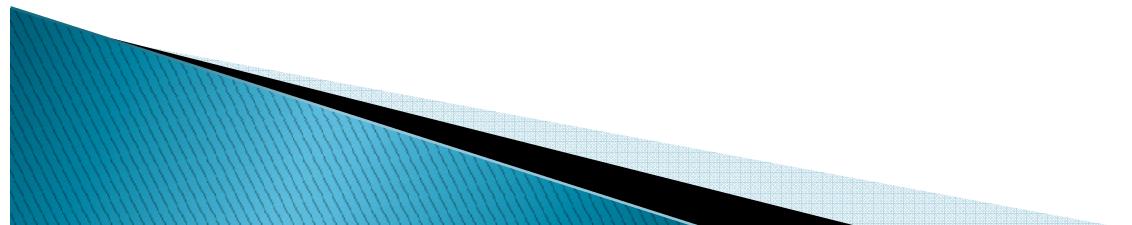
- ▶ NO\_DATA\_FOUND
- ▶ TOO\_MANY\_ROWS
- ▶ VALUE\_ERROR (erreur arithmétique)
- ▶ ZERO\_DIVIDE
- ▶ ...



# Gestion des exceptions

*Types d'erreur : Erreurs Système*

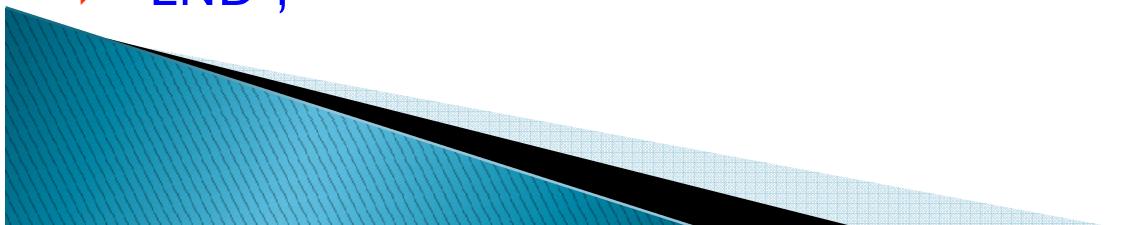
```
DECLARE
  ...
BEGIN
  ...
EXCEPTION
  WHEN NO_DATA_Found THEN traitement ;
  ...
  WHEN Too_Many_Rows THEN traitement ;
  WHEN OTHERS THEN traitement ; --optionnel
END;
```



# Gestion des exceptions

*Types d'erreur : Erreurs utilisateurs*

- ▶ On la déclare et on effectue le traitement approprié, on détecte l'anomalie au cours du programme.
  
- ▶ **DECLARE**
- ▶   **Nom\_erreur EXCEPTION ;**
- ▶ **BEGIN**
- ▶   **IF anomalie THEN RAISE nom\_erreur ;**
- ▶   **END IF ;**
- ▶   **...**
- ▶ **EXCEPTION**
- ▶   **WHEN nom\_erreur THEN traitement ;**
- ▶   **...**
- ▶ **END ;**

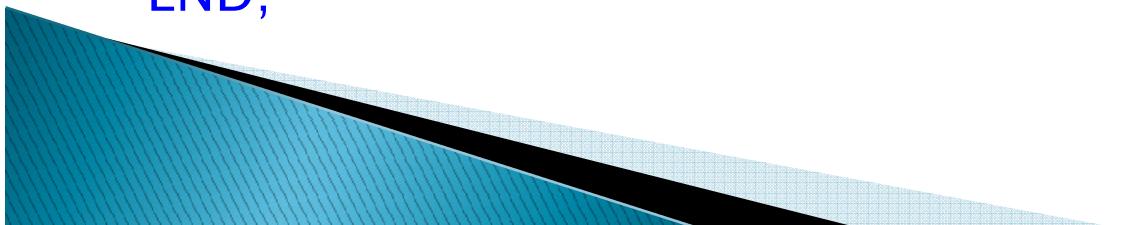


# *Reprise après exception*

Lorsqu'une exception est détectée, le traitement associé à l'exception est exécuté et ensuite on sort du bloc dans lequel l'exception a été définie.

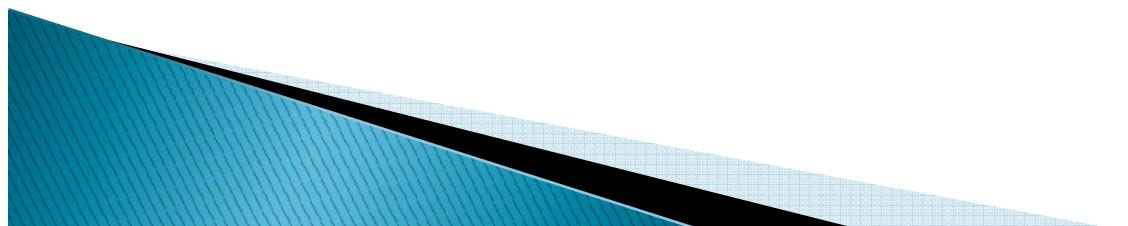
Solution possible : plusieurs blocs imbriqués.

```
BEGIN  
BEGIN  
IF exception THEN ...  
EXCEPTION  
END;  
BEGIN  
...  
END;  
END;
```



# *Reprise après exception*

- ▶ On peut donner un code à une exception :
- ▶
- ▶     **Nom\_erreur EXCEPTION ;**
- ▶     **PRAGMA EXCEPTION\_INIT(nom\_erreur,code) ;**
- ▶ Remarque : numéro pris par Oracle : -20000 à -20995
- ▶ Affichage de l'erreur :
- ▶                 ORA -xxxxx : un message



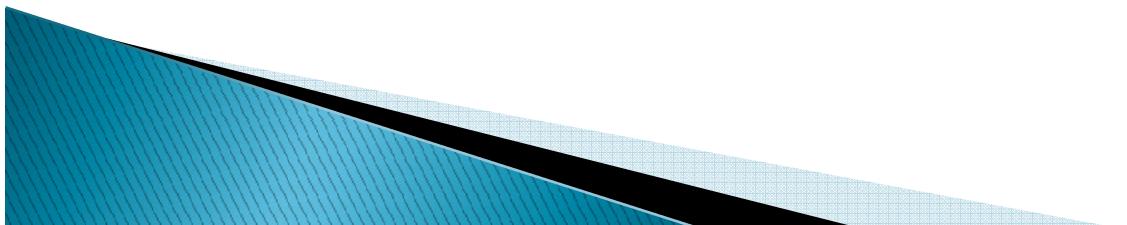
# Exemple

```
CREATE TABLE err_test  
  (nom VARCHAR2(10) PRIMARY KEY;  
   numero NUMBER CONSTRAINT no_smallnb CHECK (numero>100));
```

*INSERT INTO err\_test VALUES ("voiture",2);  
→ ORA -02290 : violation de contrainte(nom\_utl.no\_smallnb) de verification.*

*Programme PL/SQL :*

```
BEGIN  
  INSERT INTO ...  
EXCEPTION  
  IF SQLCODE=-02290 AND SQLERRM='%no_smallnb'  
    THEN  
      Affichage d'un message adapté  
    ELSE  
      Affichage des autres erreurs
```



# *Les triggers*

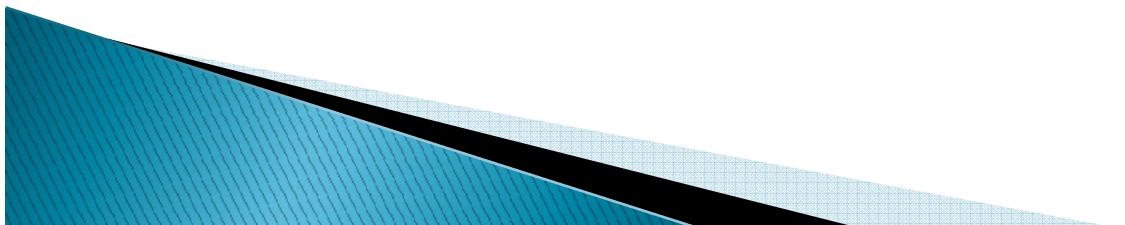
**Trigger = déclencheur, réflexe en anglais**

Un trigger lit une procédure déclenchée par des événements de mise à jour spécifiés par l'utilisateur et ne s'exécute que lorsqu'une condition est vérifiée.

```
CREATE TRIGGER nom  
BEFORE|AFTER  
    INSERT|DELETE|UPDATE  
    ON relation  
    [FOR EACH ROW]  
    [WHEN condition]
```

BEFORE/AFTER : permet de spécifier si on veut déclencher le traitement avant ou après la mise à jour.

Un trigger peut être exécuté une fois pour un seul ordre SQL, c'est ce que l'on appelle un trigger par ordre, ou à chaque ligne concernée par l'ordre, c'est ce que l'on appelle un trigger par ligne.

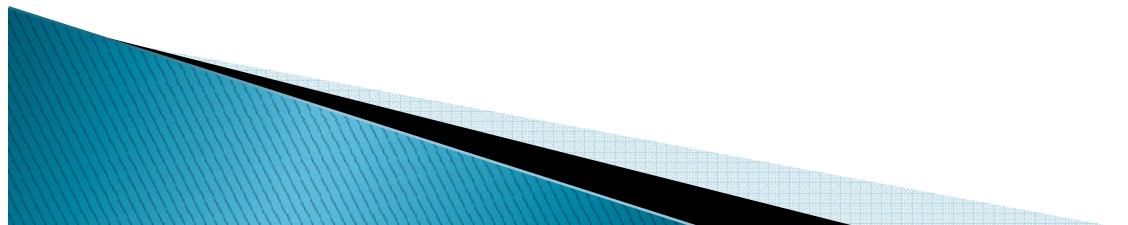


# E/S dans un programme PL/SQL

## Package DBMS\_OUTPUT

`DBMS_OUTPUT.PUT_LINE('...') ;`

`DBMS_OUTPUT.PUT_LINE('code  
erreur:', ||SQLCODE); // Concaténation`



# Exemple

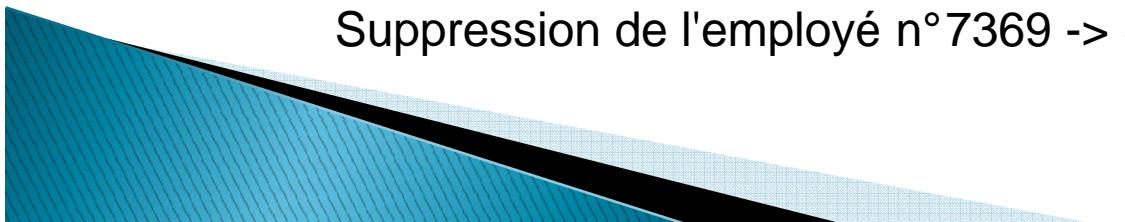
un déclencheur qui affiche le numéro et le nom d'un employé que l'on veut supprimer de la table EMP

```
CREATE OR REPLACE TRIGGER TRG_BDR_EMP BEFORE DELETE
    --avant suppression
    -- sur la table EMP
    -- pour chaque ligne
ON EMP
FOR EACH ROW
Declare
    LC$Chaine VARCHAR2(100);
Begin
    dbms_output.put_line( 'Suppression de l"employé n°' || To_char( :OLD.empno )
    || ' -> ' || :OLD.ename ) ;
End ;
```

Supprimons maintenant un employé

```
delete from emp where empno = 7369
```

Suppression de l'employé n°7369 -> SMITH



- ▶ désormais, tout nouvel employé devra avoir un numéro supérieur ou égal à 10000. Il faut donc interdire toute insertion qui ne reflète pas cette nouvelle directive

```
CREATE OR REPLACE TRIGGER TRG_BIR_EMP
BEFORE INSERT
ON EMP
FOR EACH ROW
Begin
  If :NEW.empno < 10000 Then
    RAISE APPLICATION_ERROR ( -20010, 'Numéro employé inférieur à
10000' );
  End if ;
End ;
```

Tentons d'insérer un nouvel employé avec le numéro 9999

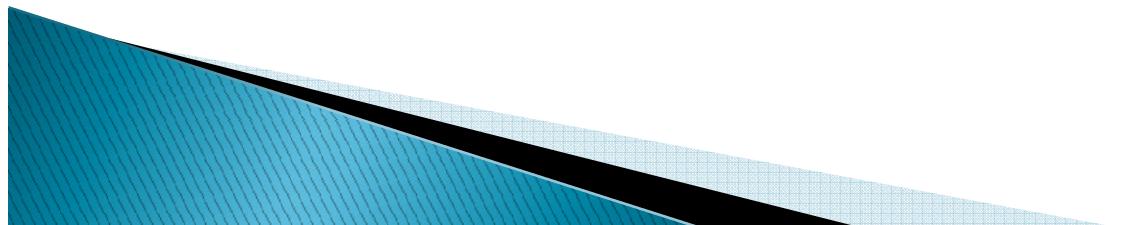
```
insert into emp (empno, ename, job) values( 9999, 'Burger', 'CLERK' ) ; insert
into emp (empno, ename, job) values( 9999, 'Burger', 'CLERK' )
```

ERREUR à la ligne 1 :

ORA-20010: Numéro employé inférieur à 10000



- ▶ CREATE OR REPLACE TRIGGER TRG\_BIUDR\_EMP BEFORE INSERT OR UPDATE OR DELETE
- ▶ -- avant insertion, modification ou suppression
- ▶ ON EMP
- ▶ FOR EACH ROW
- ▶ Begin
- ▶     If INSERTING Then
- ▶         dbms\_output.put\_line( 'Insertion dans la table EMP' ) ;
- ▶     End if ;
- ▶     If UPDATING Then
- ▶         dbms\_output.put\_line( 'Mise à jour de la table EMP' ) ;
- ▶     End if ;
- ▶     If DELETING Then
- ▶         dbms\_output.put\_line( 'Suppression dans la table EMP' ) ;
- ▶     End if ;
- ▶ End ;
- ▶

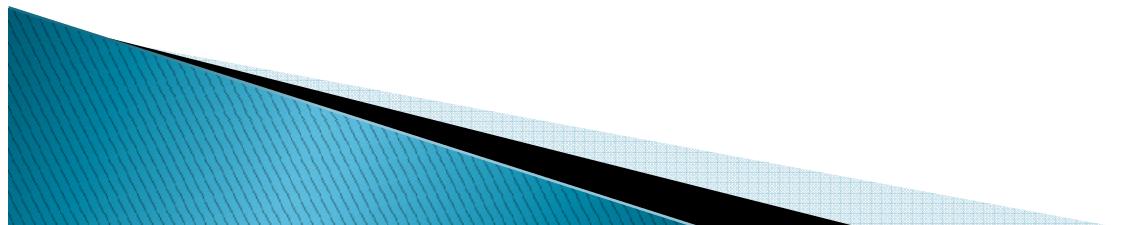


# Procédure et fonction

Bloc anonyme ou nommé

Un bloc anonyme PL/SQL est un bloc « DECLARE – BEGIN – END » comme dans les exemples précédents

Le plus souvent, on crée plutôt une procédure ou une fonction nommée pour réutiliser le code

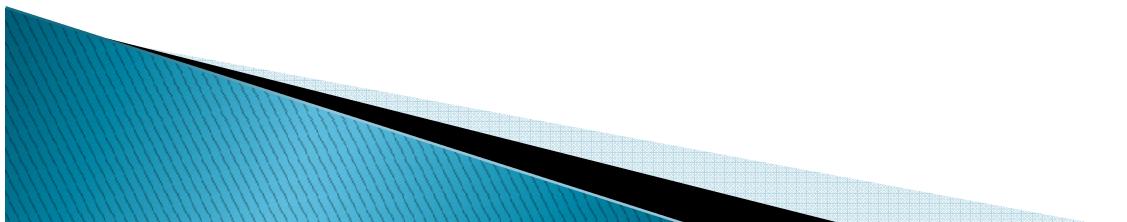


# Déclaration d'une procedure

```
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 »



# Déclaration d'une fonction

```
create or replace FUNCTION(<liste params>) RETURN <type retour>
IS
    -- déclaration des variables

BEGIN

    -- code de la fonction

END;
```

