PL/SQL :
le langage procédural d'Oracle

Un programme PLSQL :

-Un bloc anonyme

-Une fonction (renvoie un seul résultat)

- une procédure (renvoie 0 ou plusieurs résultats)

- un déclencheur (trigger) : exécution automatique si une action (définie dans le trigger) est exécutée : insert, update, delete

Bloc anonyme

DECLARE

BEGIN

instruction(s) (qui peuvent contenir des blocs imbriqués (BEGIN …END)

EXCEPTION

END

Déclaration de Variables et Constantes

variable [**CONSTANT**] type [**NOT NULL**] [:= | **DEFAULT** exp];

- Remarques :
  - Il faut initialiser les constantes et les variables qui sont déclarées **NOT NULL**.
  - Initialiser les identifiants en utilisant l'affectation (**:=**) ou le mot-clé **DEFAULT**.
  - Mettre un seul identifiant par ligne.

Quelques exemples de déclarations

```
car                 char(11);
nom_ens             varchar2(20) := 'SADEG';
val1          number;
val2                number := 20;
V3                  boolean := (val2>0);
compteur            integer := 0;
total               number(10, 2)  NOT NULL:= 0; -- initialisé
date_com            date := sysdate;
PI                  CONSTANT number (6,4) := 3.1415; -- initialisé
reussi              BOOLEAN NOT NULL := true;
Prix_ttc  number(4) := 5*200;
Date_naiss          date := TO_DATE(‘17-11-1980','DD-MM-YYYY');
```

EX1. Petit programme anonyme

```
SET SERVEROUTPUT ON -- activer l'écran

DECLARE
  rayon number (10,2) := 10.00;
  pi constant NUMBER(10,5) := 3.14159;
  surface number(10,5);

BEGIN
  surface := pi * rayon * rayon;
    DBMS_OUTPUT.PUT_LINE('Resultat : Surface = ' || surface);
  -- afficher à l'écran
END;
/
```

Résultat : Surface = 314,59

Conventions

Les noms des variables doivent être différents des noms des colonnes des tables utilisées dans un bloc

empno (est un nom de colonne) -> numero_emp (est une variable)

emp (est un nom de table), curs_emp (est un type CURSOR)

Un identifiant : <= 30 caractères, dont le premier doit être une lettre.

Type de données de base principaux

- VARCHAR2 (longueur) : <= 32767 octets
- NUMBER [(m,n)] : Numérique à virgule flottante
- DATE : de -4712 avant J.C. à +9999 après J.C.
- CHAR [(taille)]
- LONG : <= 2 147 483 647 octets
- LONG RAW : binaire <= 32 760 octets
- BOOLEAN : True, False ou NULL

Structure d’un bloc PLSQL

DECLARE -- section optionnelle

Variables,

curseurs,

exceptions définies

BEGIN -- obligatoire

Instructions SQL et PL/SQL

EXCEPTION -- section optionnelle

Actions à réaliser quand des exceptions avaient survenues dans le programme

END; -- obligatoire

/

Ex2.

```
SET SERVEROUTPUT ON
DECLARE
        chaine1 VARCHAR (5);
        chaine2 VARCHAR2(5);
BEGIN
        chaine1 := 'toto'; chaine2 := 'toto';
        IF (chaine1 = chaine2) THEN
                DBMS_OUTPUT.PUT_LINE(‘chaines égales');
        ELSE
                DBMS_OUTPUT.PUT_LINE(‘chaines différentes');
        END IF;
END;
/
```

Ex2.

/

chaines egales

PL/SQL procedure successfully completed.

SQL>

Instructions PL/SQL

Instruction Alternative

IF *condition1*

THEN *instructions1*
[**ELSIF** *condition2*
instructions2]
[ELSE *instructions3*]
END IF;

Instruction CASE

CASE *variable*
    WHEN *valeur1*
    THEN *instructions1*
    WHEN *valeur 2*
    THEN *instructions2*
     ...
    WHEN *valeur_n*
    THEN *instructions_n*
    ELSE *instructions par défaut*
END CASE;

Boucle LOOP

LOOP *instructions*

EXIT WHEN *condition* ;

END LOOP;

Boucle FOR

FOR *variable* **IN** *borne_inf* **..** *Borne_sup*

LOOP *instructions*

END LOOP;

Boucle WHILE

WHILE *condition*

LOOP *instructions*

END LOOP;

Ex3. Type structuré : exemple

```
DECLARE
TYPE point IS RECORD
    (abscisse NUMBER,
    ordonnee NUMBER
    );
    p point;

BEGIN
    p.abscisse := 5; p.ordonnee := 3;
DBMS_OUTPUT.PUT_LINE('p.abscisse = ' || p.abscisse || '     et
p.ordonnee = ' || p.ordonnee);
END;
```

Ex3. Type structuré : exemple

```
/

ABSCISSE = 10 ET ORDONNEEE = 12

PL/SQL procedure successfully completed.

SQL>
```

Affectation

Avec le symbôle d’affectation “**:=**“

ou avec

SELECT col_1, ..., col_n **INTO** v_col_1, ..., v_col_n

FROM …

EX4.

Tables :

MATIERE (nmat, nommat, coeffmat)
ETUDIANT (netud, nometud, groupetud, redouble)
NOTE (netud, nmat, moy)

EX4.

```
SET SERVEROUTPUT ON -- activer l'affichage a l'ecran
DECLARE
    nummat NUMBER;
    nommatiere VARCHAR2(30) := 'ALGO';

-- on suppose qu’une table MATIERE existe
BEGIN
    SELECT nmat INTO nummat FROM
    MATIERE WHERE nommat = nommatiere;
    DBMS_OUTPUT.PUT_LINE('Le numero de la matiere || nommatiere || '  est : ' || nummat);
END;
```

EX4.

/

Le numero de la matiere ALGO est : 2

PL/SQL procedure successfully completed.

SQL>

EX5. Référence à des types de colonnes

Tables :

PRODUIT (np, lib, coul, qs)

CLIENT (ncli, nom, adr)

ACHAT (np, ncli, qa)

EX5. Référence à des types de colonnes

DECLARE

nump PRODUIT.np%type;

-- np est une colonne de la table PRODUIT

– => nump prend le même type que celui de np

nomp PRODUIT.lib%type := *‘AGRAFEUSE'* ;

-- idem pour nomp

BEGIN

SELECT np **INTO** nump **FROM** PRODUIT **WHERE** lib = nomp;

DBMS_OUTPUT.PUT_LINE(*'Le numéro de l'article '* || nomp || *' est : '* || nump);

END;

EX5. Référence à des types de colonnes

```
/

Le numero de l'article CALCULATRICE  est : 2

PL/SQL procedure successfully completed.

SQL>
```

Référence à une ligne entière d’une table

```
DECLARE
    nom PRODUIT.nomprod%type := ‘produit_1' ;
    ligne PRODUIT%rowtype; -- ligne est du même type qu’une
    ligne de la table PRODUIT

BEGIN
SELECT * INTO ligne FROM PRODUIT WHERE
nomprod = nom;

    DBMS_OUTPUT.PUT_LINE(‘Le numéro de l''article ' ||
    ligne.nomprod || ' est :  ' || ligne.numprod);
END; /
```

Validation-Annulation

instructions
IF *erreur*
    THEN ROLLBACK; -- annulation des modifs
    ELSE COMMIT; -- validation des modifs
END;

Rmq : Si la variable d'environnement AUTOCOMMIT est positionné à ON, alors chaque instruction est validée (sinon il faut COMMIT)

Bloc EXCEPTION

```
EXCEPTION
    WHEN exeption1
    THEN  traitement
    WHEN exception2
    THEN  traitement

    WHEN OTHERS
    THEN  traitement
END;
```

Exemple d’exceptions prédéfinies

```
DECLARE
     nump NUMBER;
     nomp VARCHAR2(30) := produit_1' ;
BEGIN
         SELECT numprod INTO nump FROM PRODUIT
         WHERE nomprod = nomp;

DBMS_OUTPUT.PUT_LINE(‘Le num de l''article ' || nomp || '  est : ' || nump);


EXCEPTION
     WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE(‘pas d’’article de nom ' || nomp);
     WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE(‘il existe plusieurs articles dont le nom
                                          est ' || nomp);
     WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE(‘PROBLEME !');
END; /
```

EXCEPTION est un type

```
DECLARE
    toto EXCEPTION;

BEGIN
    IF <erreur1> THEN      RAISE toto;

EXCEPTION
    WHEN toto THEN
DBMS_OUTPUT.PUT_LINE(‘exception toto');

END; /
```

Déclaration d'une procédure

CREATE OR REPLACE PROCEDURE nomproc (arg1 type1 [T], arg2 type2 [T] , …) **IS**

déclaration des variables locales

BEGIN

instructions

END;

Où T est optionnel et vaut **IN** ou **OUT** ou **IN OUT**

Déclaration d'une fonction

CREATE OR REPLACE FUNCTION n*om_f* (arg1 type1 [T],
arg2 type2 [T], …) RETURN *type IS*

déclaration des variables locales

BEGIN

instructions

END;

Où T est optionnel et vaut **IN** ou **OUT** ou **IN OUT**

Passage des paramètres

- IN : passage par valeur
- OUT : aucune valeur passée, sert de valeur de retour
- IN OUT : passage de paramètre par référence

Rmq : par défaut, le passage se fait par valeur IN fait de type IN.

Exemple

```
CREATE OR REPLACE PROCEDURE increment_3
   (val IN OUT NUMBER)
IS
BEGIN
          val := val + 3;
       DBMS_OUTPUT.PUT_LINE('val = ' || val)
END;
```

```
CREATE OR REPLACE FUNCTION plus_grand
    (a NUMBER, b NUMBER) RETURN NUMBER IS

BEGIN

    IF a > b THEN RETURN a;
        ELSIF b > a  THEN RETURN b;
        ELSE RETURN 0;
    END IF;

END
```

Utilisation

SELECT plus_grand(121, 102) **FROM** DUAL;

Accès aux tables d'une BD

```
SQL> SET SERVEROUTPUT ON

SQL> DECLARE
                v_nom emp.ename%TYPE; -- même type que la colonne ename
        BEGIN
            SELECT ename INTO v_nom FROM emp
           WHERE empno=7845; -- un numéro d'employé

        DBMS_OUTPUT.PUT_LINE(' Le nom est ' || v_nom);
END;
/

PL/SQL Procedure successfully completed.
```

Plusieurs variables hôtes

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
        v_nom          emp.ename%TYPE;
        v_salaire      emp.sal%TYPE;
     BEGIN
       SELECT   ename,sal        INTO v_nom,v_salaire
       FROM emp WHERE ROWNUM = 1; -- num de ligne

        DBMS_OUTPUT.PUT_LINE('NOM : ' || v_nom);
        DBMS_OUTPUT.PUT_LINE(' SALAIRE :' || v_salaire);
     EXCEPTION
           WHEN OTHERS
                THEN NULL;
     END;
/
```

Variables de substitution de SQL*Plus

```
SQL> SELECT * FROM DEPT WHERE DEPTNO = '&1';
Enter value for 1: 10
old   1: SELECT * FROM DEPT WHERE DEPTNO = '&1'
new   1: SELECT * FROM DEPT WHERE DEPTNO = '10'

    DEPTNO     DNAME          LOC
---------- ------------- ------------------------------
        10      ACCOUNTING      NEW YORK
```

Variables hôtes de SQL*Plus

```
SQL> VARIABLE x NUMBER
SQL> BEGIN
  2  SELECT SUM(sal) INTO :x FROM EMP; -- variable hôte
  3  END;
  4  /
PL/SQL procedure successfully completed.
SQL> PRINT x
         X
----------
     29025
```

Passage de valeurs à un script

1.

```
SQL> START fichier.sql valeur1 valeur2 …
```

Exemple

```
SQL> SELECT * FROM DEPT WHERE DEPTNO = '&1';

SQL> SAVE question1.sql


SQL> START question1.sql     20
old   1: SELECT * FROM DEPT WHERE DEPTNO = '&1'
new   1: SELECT * FROM DEPT WHERE DEPTNO = '20'


    DEPTNO     DNAME          LOC
---------- -------------- -------------
        20         RESEARCH       DALLAS
```

Instructions de modification de tables dans PL/SQL

Modifier le contenu des tables à l'aide des ordres du LMD

- INSERT
- UPDATE
- DELETE

```
DECLARE
   v_deptno emp.deptno%TYPE :=10;
BEGIN
   UPDATE DEPT SET loc = ‘LE HAVRE'
   WHERE deptno = v_deptno;
   COMMIT;
END;
/
```

Exemple 1 : bloc anonyme

```
SET SERVEROUTPUT ON
DECLARE
   v_sal emp.sal%TYPE;
BEGIN
   SELECT sal INTO v_sal
    FROM emp WHERE empno = '7876';
   IF v_sal BETWEEN 3000 AND 5000 THEN
         DBMS_OUTPUT.PUT_LINE('gros salaire') ;
   ELSIF v_sal BETWEEN 1500 AND 2999 THEN
         DBMS_OUTPUT.PUT_LINE('moyen salaire') ;
   ELSE
         DBMS_OUTPUT.PUT_LINE('petit salaire') ;
   END IF;
END;
/
```

Exemple avec CASE

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
     2 v_sal  emp.sal%TYPE; v_sal a le même type que sal
3 BEGIN
     4    SELECT sal  INTO  v_sal  FROM emp
5    WHERE empno = &numero;-- il demande un numero
6    CASE
     7    WHEN  v_sal  BETWEEN  4000  AND  4999  THEN
     8         DBMS_OUTPUT.PUT_LINE(‘salaire élevé') ;
     9    WHEN  v_sal  BETWEEN  2000  AND  2999  THEN
     10        DBMS_OUTPUT.PUT_LINE(‘salaire moyen') ;
     11   ELSE
     12        DBMS_OUTPUT.PUT_LINE(‘salaire faible') ;
     13 END CASE;
14 END;
15 /
Enter value for matricule: 7839
```

Curseurs PLSQL

Curseurs

- On utilise un curseur pour traiter les relations ligne par ligne et/ou quand on ignore combien de lignes seront rapportées par une requête.
- Le serveur utilise une zone locale (tampon) pour exécuter les instructions SQL et pour stocker les informations en cours de traitement
- Il y a des curseurs implicites et des curseurs explicites
- Les curseurs explicites sont définis par le programmeur
- Un curseur possède des attributs

Curseurs

Exemple de curseur correspondant à l'ordre:
Select empno, ename job from emp;

7369 SMITH CLERK
7566 JONES MANAGER
7788 SCOTT ANALYST
7876 ADAMS CLERK
7902 FORD ANALYST
etc …

Déclaration d'un Curseur

DECLARE

CURSOR *cursor_name* IS

instruction_select **;**

Exemple :

```
DECLARE
    CURSOR c_emp IS
        SELECT empno, ename FROM emp
        WHERE sal > 3000;
```

```
DECLARE
    CURSOR emp_cur IS  SELECT * FROM EMP;
    ligne emp_cur%rowtype  - - ligne de type curseur emp_cur

BEGIN
    OPEN emp_cur;
    LOOP
        FETCH emp_cur INTO ligne;
        EXIT WHEN emp_cur%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('nom = ' || ligne.ename);
    END LOOP;
    CLOSE emp_cur;
END;
```

Table :

PERSONNE(numpers, nom, prenom)
ENFANT(numenf, nom_enf, prenom_enf, pere, mere)

```
 Create table personne
  (numpers integer primary key,
  Nom  varchar(20) not null,
  Prenom varchar(20)
);

Create table enfant
  (numenf integer primary key,
  Prenom_enf varchar(20),
  Pere integer not null references personne(numpers),
  mere integer not null references personne(numpers) );
```

```
Insert into personne values (1, 'DAMA', 'MARC') ;
Insert into personne values (2, 'DAMA' 'LAURE') ;
Insert into personne values (3, 'LELLOUC', 'ERIC') ;
Insert into personne values (4, 'LARTI', 'MARIE') ;

Insert into enfant values (10, 'JACQUES',1, 2 ) ;
Insert into enfant values (11, 'ALINE', 1, 2) ;
Insert into enfant values (12, 'ZOE', 1, 2) ;
Insert into enfant values (13, 'CLAIRE', 1, 2 ) ;
Insert into enfant values (14, 'ZACHARY', 1, 2) ;
Insert into enfant values (15, 'ARTHUR', 3, 4) ;
Insert into enfant values (16, 'MARY', 3, 4) ;
```

Curseur paramétré

```
set serveroutput on
declare
 CURSOR parent IS SELECT * FROM PERSONNE;
 p PERSONNE%ROWTYPE;
 CURSOR enfant(numparent integer) IS
    select * from enfant where pere = numparent or mere = numparent;
 e ENFANT%ROWTYPE;
BEGIN
  FOR p in parent
  LOOP
   DBMS_OUTPUT.PUT_LINE('Les enfants de ' ||  p.prenom || '  ' || p.nom ||
' sont : ');
     FOR e IN enfant(p.numpers)
     LOOP
      DBMS_OUTPUT.PUT_LINE(' *  ' || e.prenom_enf );
     END LOOP;
    END LOOP;
END;
/
```

Autres exemples

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2   rec emp%ROWTYPE;
  3    CURSOR c1 IS SELECT * FROM emp
  4                 ORDER BY sal DESC;
  5  BEGIN
  6   OPEN c1;
  7    FOR ind IN 1..5 – on sait qu'il y a au moins 5 tuples
  8     LOOP
  9          FETCH c1 INTO rec;
 10          DBMS_OUTPUT.PUT_LINE(rec.ename || ' ' ||
 11                                TO_CHAR(rec.sal));
 12    END LOOP;
 13    CLOSE c1;
 14  END;
 15  /
KING 5000
SCOTT 3000
…
BLAKE 2850
```

```
SQL> DECLARE
  2      CURSOR curs IS SELECT * FROM emp
  3              ORDER BY desc; -- par salaires décroissants
  4      enreg emp%ROWTYPE;
  5  BEGIN
  6   OPEN curs;
  7   LOOP
  8         FETCH curs INTO enreg;
  9         EXIT WHEN curs%ROWCOUNT > 5 OR curs%NOTFOUND ;
 10              DBMS_OUTPUT.PUT_LINE(enreg.ename || ‘        ' || enreg.sal);
 12         END LOOP;
 13         CLOSE curs;
 14  END;
 15  /

KING        5000
SCOTT       3000
...
```

Quelques attributs d'un Curseur

Attribut	Type	Description
%ISOPEN	BOOLEAN	TRUE si curseur ouvert.
%NOTFOUND	BOOLEAN	TRUE si le dernier fetch n'a ramené aucune ligne.
%FOUND	BOOLEAN	TRUE tant que Fetch ramène des lignes.
%ROWCOUNT	NUMBER	Nombre total de lignes ramenées par Fectch.

Curseur Enregistrement (Record)

```
DECLARE
        CURSOR c1 IS
         SELECT e.ename AS nome, d.loc AS nomv
           FROM emp e, dept d
          WHERE e.deptno = d.deptno
            AND d.deptno = 10;
enreg  c1%ROWTYPE;
BEGIN
        OPEN c1;
        LOOP
            FETCH c1 INTO enreg; -- curseur dans l'enregistrement enreg
            EXIT WHEN c1%NOTFOUND; -- fin, si plus de lignes
            DBMS_OUTPUT.PUT_LINE('nom :  ' || enreg.nome||'    Localisation : '||
enreg.nomv);
   -- affichage de chaque ligne de la table
        END LOOP;
        CLOSE c1;
END;
/
```

Utilisation des attributs du curseur

```
SQL> DECLARE
  2    CURSOR c1 IS SELECT e.ename, e.sal FROM emp e
  3                 ORDER BY e.sal DESC;
  4    enreg       c1%ROWTYPE; -- enreg est de type curseur c1
  5    BEGIN
  6     IF NOT c1%ISOPEN  THEN
  7                 OPEN c1;
  8     END IF;
  9      WHILE (c1%ROWCOUNT < 5)   LOOP
 10         FETCH c1 INTO enreg;
 11         EXIT WHEN c1%NOTFOUND;
 12          DBMS_OUTPUT.PUT_LINE(‘ NOM : ‘ || enreg.ename ||'   SAL :  ‘ '|| enreg.sal);
 13       END LOOP;
 14       IF c1%ISOPEN THEN
 15            CLOSE c1;
 16       END IF;
 17  END;
 18  /
KING 5000
SCOTT 3000
FORD 3000
```

Curseur dans une boucle FOR

Le ‘FOR’ Simplifie l'usage d'un curseur

Ouverture, extraction et fermeture implicite du curseur

L’enregistrement (RECORD) est également déclaré

```
    FOR nom_enreg IN nom_curseur
-- pas besoin de déclarer nom_enreg
LOOP
                instructions
    END LOOP;
```

Un exemple

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2                 FOR nomdep IN (SELECT * FROM dept)
  3                 LOOP
  4                           DBMS_OUTPUT.PUT_LINE(nomdep.dname);
  5                 END  LOOP;
  6  END;
  7 /
ACCOUNTING
RESEARCH
SALES
OPERATIONS

PL/SQL procedure successfully completed.
```

Curseur et enregistrement

```
DECLARE
     CURSOR c1 IS
                SELECT * FROM DEPT;
BEGIN
  FOR var1 IN c1 - - on ne déclare pas var1
  LOOP
               DBMS_OUTPUT.PUT_LINE('Nom dep : ' ||
var1.dname);
  END LOOP;
END;
/
ACCOUNTING
…
```

Boucle WHILE - Exemple

```
SQL> DECLARE
  2       ligne scott.emp%ROWTYPE;
  3       CURSOR c1 IS SELECT * FROM emp
  4       ORDER BY sal DESC;
  5  BEGIN
  6    OPEN c1;
  7    WHILE (c1%ROWCOUNT < 5) LOOP
  8          FETCH c1 INTO ligne;
  9          DBMS_OUTPUT.PUT_LINE(l'Nom : ' ||  ligne.ename || '  Sal : ' ||
 10                                    TO_CHAR(ligne.sal));
 11    END LOOP;
 12    CLOSE c1;
 13  END;
 14  /

Nom : KING  Sal : 5000
Nom : SCOTT  Sal : 3000
…
Nom : BLAKE  Sal : 2850
```

Types de Données Composés

Exemple

```
SQL> DECLARE
  2     TYPE emp_record_type IS RECORD
  3        ( ename    VARCHAR2( 25 ),
  4          job         VARCHAR2( 25 ),
  5          sal   NUMBER( 7,2 )
  6        );
  7     employee_record    emp_record_type;
  8     BEGIN
  9        SELECT ename, job, sal -- une seule ligne!
 10          INTO employee_record
 11          FROM emp
 12         WHERE empno = 7839;
 13     DBMS_OUTPUT.PUT_LINE(employee_record.ename);
 14     DBMS_OUTPUT.PUT_LINE(employee_record.job);
 15     DBMS_OUTPUT.PUT_LINE(employee_record.sal);
 16  END;
 17  /
KING PRESIDENT 5000
```

Exemple

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2       enreg emp%ROWTYPE;
  3  BEGIN
  4       SELECT * INTO enreg FROM emp
  5       WHERE ROWNUM = 1;
  6       DBMS_OUTPUT.PUT_LINE(enreg.ename);
  7       DBMS_OUTPUT.PUT_LINE(enreg.sal);
  8  END;
  9   /
SMITH
800

PL/SQL procedure successfully completed.
```

Traitement des erreurs
EXCEPTIONS

- Une Exception est une erreur PL/SQL déclenchée pendant l'exécution du code
- Deux modes
  - Implicite : une erreur Oracle est apparue.
  - Explicite : par l'utilisateur.
- Gestion : Section **EXCEPTION**

Les Exceptions

DECLARE
…
Exception_1 EXCEPTION;

BEGIN ….
If …. THEN **RAISE exception_1;**

EXCEPTION - - bloc de gestion des exceptions
WHEN *exception_1* [OR *exception2* . . .] **THEN**
***inst11*;**
***inst22*;**
. . .
[WHEN *exception3* [OR *exception4* . . .] THEN
inst111;
inst222;
. . .]
[WHEN OTHERS THEN
inst1111;
inst2222;
. . .]

Remarques

1. Le mot-clé **WHEN OTHERS** gère toutes les erreurs qui ne sont pas encore gérées.

2. **WHEN OTHERS** est la dernière clause du bloc EXECPTION.

Exemple d'exception prédéfinie

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     nb NUMBER;
  3  BEGIN
  4     nb := ‘chaine'; -- affectation incorrecte !
  5     DBMS_OUTPUT.PUT_LINE(‘OK');
  6  EXCEPTION
  7     WHEN VALUE_ERROR THEN
  8         DBMS_OUTPUT.PUT_LINE( ‘Erreur : affectation incorrecte');
  9  END;
 10  /
```

Erreur : affectation incorrecte

PL/SQL procedure successfully completed.

Quelques exceptions

- NO_DATA_FOUND : un curseur ne ramène pas de lignes
- TOO_MANY_ROWS : un curseur ramène plusieurs lignes
- INVALID_CURSOR : curseur invalide
- ZERO_DIVIDE : division par zéro
- DUP_VAL_ON_INDEX : valeur dupliquée

Exceptions définies par l’utilisateur

- Déclarées dans la section **DECLARE**
- On les déclenche dans un bloc par l'instruction **RAISE**
- Dans la section **EXCEPTION**, référencer le nom de l’exception (défini dans la section **DECLARE**)
- Cette exception ne peut être appelée que durant l'exécution d'un sous-programme stocké dans le serveur de données

Exemple

```
SQL> DECLARE
  2     CODE_DEP NUMBER(4);
  3     DEPTNO_INCORRECT EXCEPTION;
  4     CURSOR c1 IS SELECT deptno FROM dept WHERE ROWNUM = 1;
  5  BEGIN
  6     OPEN c1;
  7     FETCH c1 INTO CODE_DEP;
  8     CLOSE c1;
  9     IF CODE_DEP NOT IN (10,20,30) THEN
 10         RAISE DEPTNO_INCORRECT;
 11     END IF;
 12     DBMS_OUTPUT.PUT_LINE(‘Numéro de departement OK !');
 13  EXCEPTION
 14     WHEN DEPTNO_INCORRECT THEN
 15       DBMS_OUTPUT.PUT_LINE('Code du departement invalide');
             DBMS_OUTPUT.PUT_LINE(‘ SQLCODE = ' || SQLCODE || ' ' ||
 16        ‘ SQLERRM = ' || SQLERRM);
- - sqlcode et sqlerrm : voir transparent suivant
 16  END;
 17  /
```

Messages et codes d’erreurs

- SQLCODE : retourne la valeur numérique correspondant au code de l’erreur d’Oracle.

- SQLERRM : retourne le message associé au numéro d’erreur