Ingénierie des bases de données
Chapitre 4 : Le langage PL/SQL

Dr. Mariem MAHFOUDH & Dr. Ines Zouari

mariem.mahfoudh@gmail.com, ines.zouari@isims.usf.tn

2 LSI-ADBD, ISIMS, 2023-2024

Bibliographie

Ce cours a été construit en se basant sur les références suivantes :

- Livre "PL/SQL Initiation et maîtrise", Pr. Jamel Feki
- Cours "PL/SQL", Dr. Malika SMAIL-TABBONE
- Cours "Système de gestion de base de données II", Dr. Lotfi Bouzguenda
- Rapport "Oracle PL/SQL", Tellora

Plan

Plan

Introduction

PL/SQL

- Procedural Language / Structured Query Language (PL/SQL) est un langage de programmation pour SQL.
- Extension procédurale du langage SQL.
- Développement d'applications complexes autour de BD :
  - Structures de contrôle (conditionnelles, itérations) ;
  - Éléments procéduraux (procédures, fonctions).

Principaux objectifs de PL/SQL

- Enchaîner plusieurs instructions SQL.
- Augmenter l'expressivité de SQL.
- Optimiser l'exécution d'ensemble de commandes SQL.
- Réutiliser le code des programmes.

Plan

Structure d'un bloc PL/SQL

Un programme PL/SQL est structuré comme suit :

```
DECLARE
        ----------
        ----------
BEGIN
        ----------
        ----------
EXCEPTION
        ----------
        ----------
END ;
```

- La zone DECLARE est optionnelle et sert à la déclaration des variables, des constantes, ou des curseurs.
- La zone BEGIN constitue le corps du programme.
- La zone EXCEPTION est optionnelle et permet de préciser les actions à entreprendre lorsque des erreurs sont rencontrées.
- Le END répond au BEGIN précédent, il marque la fin du script.

Structure d'un bloc PL/SQL

- Commandes de SQL utilisables dans un bloc PL/SQL
  - Toutes les commandes de SQL/LMD (SELECT, INSERT, UPDATE...).
    Attention ! forme particulière de la commande SELECT (SELECT...INTO...).
  - La partie gestion de transactions (commit, rollback, savepoint).
  - Les commandes LDD ne sont pas utilisables dans les blocs PL/SQL (create table, create view, create index, drop table...).
- Instructions spécifiques à PL/SQL :
  - Définition de variables ;
  - Traitements conditionnels ;
  - Traitements répétitifs ;
  - Traitement des curseurs ;
  - Traitement des erreurs.
- Commentaires dans un bloc PL/SQL
  - - - ceci est un commentaire sur une ligne
  - /* ceci est un commentaire long sur plusieurs lignes */

Structure d'un bloc PL/SQL

Exemple d'un bloc PL/SQL

```
DECLARE
    qté_en_stock NUMBER(4);
BEGIN
    SELECT quantité INTO qté_en_stock FROM inventaire WHERE
nom_produit = 'RAQUETTE TENNIS';
   IF qté_en_stock > 0 THEN
       UPDATE inventaire SET quantité = quantité-1 WHERE
nom_produit = 'RAQUETTE TENNIS';
       INSERT INTO ventes VALUES ('vente raquette tennis',
SYSDATE);
    ELSE INSERT INTO ventes VALUES ('rupture stock
raquette tennis', SYSDATE);
    END IF;
    COMMIT;
END;
```

Plan

Affichage à l'écran

- Les procédures du package DBMS_OUTPUT permettent d'écrire/lire des lignes dans un tampon (buffer) depuis un bloc PL/SQL ou une procédure.
- Ces lignes peuvent être affichées à l'écran (sortie standard).
- Par défaut, les fonctions d'affichage sont désactivées. Il convient, à moins que vous ne vouliez rien voir s'afficher, de les activer avec la commande SET SERVEROUTPUT ON.
- Procédures d'écriture (dans le buffer ou à l'écran)
  - PUT : ajout d'un texte sur la ligne courante
  - NEW_LINE : ajout d'un retour à ligne
  - PUT_LINE : put + new_line

Affichage à l'écran

Exemple d'affichage Hello World ! avec PL/SQL

```
SET SERVEROUTPUT ON
DECLARE
    c varchar2 (15) := ' Hello World ! ' ;
BEGIN
    DBMS_OUTPUT. PUT_LINE ( c ) ;
END;
```

Plan

Variables et constantes

- On doit déclarer des variables et des constantes avant de les utiliser dans un bloc, dans la partie DECLARE.
- Lors de l'exécution du bloc, la valeur d'une variable peut changer contrairement à la valeur d'une constante.
  nom_variable type [[NOT NULL] [DEFAULT—:= expression];
  nom_constante CONSTANT type := valeur;
- On peut affecter une valeur initiale ou une valeur par défaut à une variable. Dans le cas contraire, la variable est initialisée à NULL.
- L'affectation d'une valeur à une constante est obligatoire.
- Identificateurs Oracle :
  - 30 caractères au plus ;
  - doit commencer par une lettre ;
  - peut contenir lettres, chiffres, _, $ et # ;
  - Pas sensible à la casse ;
  - Ne doit pas être un mot réservé.

Variables

Exemple

```
DECLARE
qty_in_stock NUMBER(4);
birthday DATE;
employees_count SMALLINT := 0;
pi CONSTANT REAL := 3.14159;
radius REAL := 1;
area REAL := pi*radius**2;
groupe_sanguin CHAR NOT NULL DEFAULT 'O';
heures_travail INTEGER DEFAULT 40;
credit_maximum CONSTANT REAL := 500.00;
BEGIN
```

Il est important de noter qu'une seule déclaration par ligne est permise

Expressions dans PL/SQL

- Opérateurs arithmétiques

  `+ , - , / , * ,**`

  * : effectue la multiplication.
  ** : effectue une opération exponentielle.

- Opérateur de concaténation de chaînes

  || Ex. 'a' || 'b' || 'c' || 'd' donne 'abcd'

- Opérateurs de comparaisons

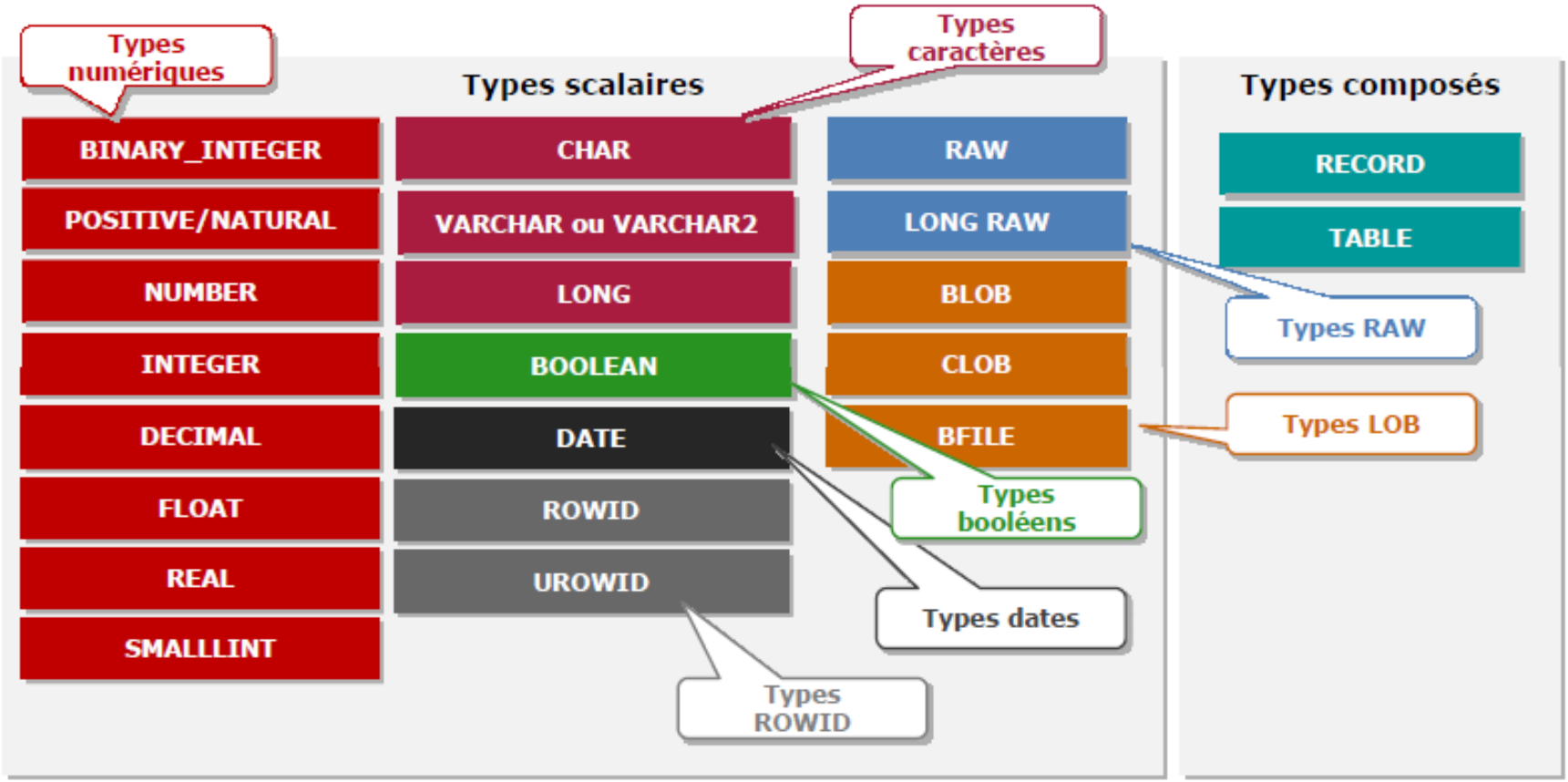
  `= , < , > , <= , >= , <> , IS NULL, LIKE, BETWEEN, IN`

- Opérateurs logiques

  `AND, OR, NOT`

Types de données

Les types PL/SQL se répartissent en deux classes : scalaire et composé.



Types de données

Types scalaires

- PL/SQL supporte les types du langage SQL (Char, Varchar2, Number, Date) et offre des types supplémentaires pour déclarer des variables et des constantes.
- ROWID : un ROWID est un identificateur interne unique de chaque n-uplet dans la base.
- Types LOB (Large OBjects) : pointer sur des objets larges et les enregistre séparément, comme les images, les vidéos, etc. BFILE, BLOB, CLOB, NCLOB.
- Types référence REF CURSOR, REF type_objet.

Types de données

L'attribut %TYPE

Il est possible de déclarer une variable par référence à une variable ou à une colonne d'une table de la base de données en utilisant l'attribut %TYPE qui se lit "de même type que".

Syntaxe :

NomVariable nomTable.nomColonne%TYPE

Exemple :

Soit la table Laboratoire(labno, labnom)

DECLARE
V_Labno Laboratoire.labno%TYPE

- - V_Labno prend le type de la colonne labno de la table Laboratoire.

L'attribut %ROWTYPE

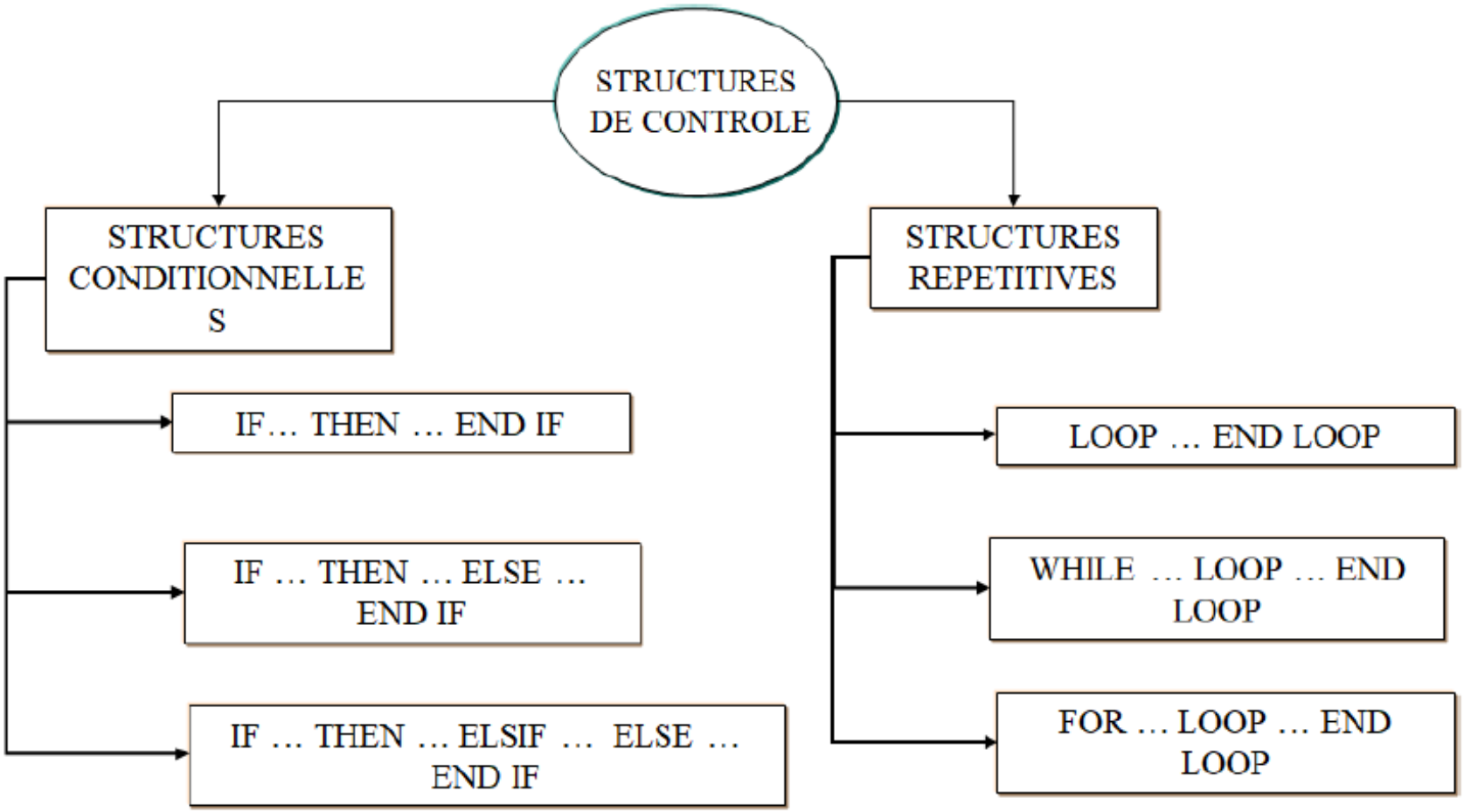
%ROWTYPE : attribut fournissant le type d'un tuple d'une table, d'une vue ou d'un curseur (type record (enregistrement)
Il permet de déclarer un enregistrement PL/SQL qui représente une ligne dans une table de base de données, sans lister toutes les colonnes.

Exemple

```
DECLARE
employee_rec EMPLOYEES%ROWTYPE
BEGIN
SELECT * INTO employee_rec FROM EMPLOYEES
WHERE emp_id = 123;
IF (employee_rec.manager_id IS NULL) THEN
    dbms_output.put_line ('salarié 123 sans chef');
END;
```

Plan

Structures de contrôle



Structures de contrôle

Conditionnelle (IF)

IF condition
THEN commandes;
[ELSE commandes ;]
END IF;

Exemple

```
BEGIN
IF x>y THEN max := x END IF;


IF nb_ventes > quota THEN
    UPDATE Ventes set ... WHERE ..;
ELSE
    DBMS_output.putline("Quota insuffisant");
END IF;
END;
```

Conditionnelle (ELSIF)

```
Declare
X number(3);
Y varchar(20)
Begin
IF X > 0 THEN
    Y := 'POSITIF';
ELSIF X < 0 THEN
    Y := 'NEGATIF';
ELSE
    Y := 'NUL';
END IF;
End;
```

Conditionnelle multiple (CASE)

CASE expression
WHEN valeur THEN commandes;
WHEN valeur THEN commandes;
...
[ELSE commandes ;]
END CASE;

Exemple

```
CASE note
WHEN 'A' THEN dbms_output.put_line ('bon');
WHEN 'B' THEN dbms_output.put_line ('moyen');
WHEN 'C' THEN dbms_output.put_line ('médiocre');
ELSE dbms_output.put_line ('note inexistante');
END CASE;
```

Structures de contrôle

Boucle "infinie" (LOOP)

L'instruction LOOP est un type spécial d'instruction de bouclage, car elle ne comporte pas de clause de condition de fin.

```
[<<nom_boucle>>]
LOOP
commandes;
END LOOP;
Pour sortir de la boucle :
EXIT [<<nom_boucle>>] [WHEN condition];
```

Incrémenter jusqu'à la valeur 10 un nombre initialisé à 0

```
val:=0;
LOOP
    val:=val+1;
    IF (val=10) THEN
       EXIT;
    END IF;
END LOOP;
```

Boucle " tant-que" (WHILE)

```
[<<nom_boucle>>]
WHILE condition LOOP
commandes;
END LOOP [nom_boucle];
```

Exemple

```
DECLARE
X NUMBER(3) := 1;
BEGIN
WHILE X <= 100 LOOP
    X := X+2;
END LOOP;
END;
```

Boucle "pour" (FOR)

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

Exemple

```
<<boucle>>
FOR i IN 1..100 LOOP
    dbms_output.put_line (i);
END LOOP boucle;


/* 1, 2, 3 ... 100 */
```

Boucle for

```
<<boucleInverse>>
FOR i IN REVERSE 1..100 LOOP
dbms_output.put_line(i)
END LOOP boucleInverse;


/* 100, 99, 98, ..*/
```

Plan

Curseurs

- Un curseur est une sorte de pointeur permettant de parcourir le résultat d'une requête tuple par tuple.
- Deux types de curseurs à distinguer :
  - curseur implicite : créé et géré par le SGBD à chaque ordre SQL
  - curseur explicite : créé et géré par l'utilisateur afin de pouvoir traiter un SELECT qui retourne plusieurs lignes
- L'utilisation d'un curseur explicite nécessite 4 étapes :
  1. Déclaration du curseur (CURSOR IS)
     - on associe une requête SELECT au curseur
     - aucun effet visible
  2. Ouverture du curseur (OPEN)
     - la requête SELECT est évaluée
     - le curseur pointe vers le premier tuple
  3. Lecture du tuple courant et passage au tuple suivant (FETCH)
  4. Fermeture du curseur (CLOSE)

Déclaration d'un curseur

- Déclaration de curseur (dans la partie déclarative d'un bloc)
  CURSOR nom_curseur[(argument1 type:=valeur, ...)] IS requête;

Exemple

```
DECLARE
CURSOR c1
IS SELECT numProduit, libellé
FROM produit ORDER BY numProduit;
CURSOR c2 (arg1 Produit.numProduit%TYPE := 5)
IS SELECT numProduit, libelle
FROM produit WHERE numProduit > arg1; -- arg1 vaut 5 par
défaut
```

Ouverture d'un curseur

- Ouverture d'un curseur (dans la partie exécutable d'un bloc)
  OPEN nom_curseur [(valeur_argument1 , ...)];

```
DECLARE
CURSOR C1 IS SELECT numProduit, libelle
FROM produit ORDER BY numProduit;
CURSOR C2 (arg1 Produit.numProduit%TYPE := 5) IS SELECT
numProduit, libelle
FROM produit WHERE numProduit > param1;
BEGIN
OPEN C1;
OPEN C2 (10);
```

Lecture du tuple courant

- Lecture du tuple courant et passage au tuple suivant
  FETCH nom_curseur INTO nom_variable_type_record; ou
  FETCH nom_curseur INTO nom_var1, nom_var2 ...;
- Attributs d'un curseur : nom_curseur%ATTRIBUT
  - %FOUND : retourne (True) si un tuple a été trouvé (par FETCH)
  - %NOTFOUND : retourne (True) si aucun tuple trouvé
  - %ISOPEN : retourne (True) si le curseur est déjà ouvert
  - %ROWCOUNT : nombre de tuples déjà traités

Fermeture de curseur

- Fermeture explicite d'un curseur
  CLOSE nom_curseur;
- Parcours des tuples d'un curseur à l'aide d'une boucle FOR
  - - ouverture implicite
  FOR variable_record IN nom_curseur LOOP
  - - disposer du tuple courant
  END LOOP ;
  - - fermeture implicite du curseur

Curseurs

Exemple

Soit la table Produit (numProduit, libelle, prix, numFournisseur)

```
DECLARE
CURSOR C1 (no_four Produit.numFournisseur%TYPE)
IS SELECT libelle, prixUnitaire
FROM Produit
WHERE numFornisseur = no_four;
un_produit C1%ROWTYPE ;
BEGIN
OPEN C1(123);
IF C1%ISOPEN THEN FETCH C1 INTO un_produit;
WHILE C1%FOUND LOOP
DBMS_OUTPUT.PUT_LINE ('Le produit : ' ||
un_produit.libelle|| ' coûte: ' ||un_produit.prix);
FETCH C1 INTO un_produit;
END LOOP;
ELSE DBMS_OUTPUT.PUT_LINE ('erreur lors de lóuverture du
curseur');
END IF; CLOSE C1;
END;
```

Même bloc avec parcours du curseur avec une boucle FOR

```
DECLARE
CURSOR CUR (no_four Produit.numFournisseur%TYPE)
IS SELECT libelle, prixUnitaire
FROM Produit
WHERE numFornisseur = no_four; un_produit CUR%ROWTYPE ;
BEGIN
FOR un_produit IN CUR(123) LOOP
DBMS_OUTPUT.PUT_LINE (un_produit.libelle|| ' coûte: '
||un_produit.prix);
END LOOP;
END;
```

Mise à jour à travers un curseur

- Déclaration d'un curseur en vue d'une mise à jour
  CURSOR nom_curseur IS requête FOR UPDATE;
- Modification ou suppression du tuple courant désigné par le curseur
  UPDATE nom_table ... WHERE CURRENT OF nom_curseur;
  DELETE nom_table WHERE CURRENT OF nom_curseur;

Plan

Gestion des exceptions

- Les exceptions sont des erreurs liées à notre programme qu'on gère de telles façons à ne pas avoir une erreur d'exécution dans notre programme.
- Le mécanisme des exceptions est implémenté dans la plupart des langages récents, notamment orientés objet.
- Cette façon de programmer a quelques avantages immédiats :
  - obliger les programmeurs à traiter les erreurs. Le but est de vous assurer que vous n'avez pas oublié d'erreur.
  - Rattraper les erreurs en cours d'exécution. Certaines erreurs d'éxecution sont rattrapables, autrement dit, il est possible de résoudre le problème sans interrompre le programme.
  - Écrire le traitement des erreurs à part. Pour des raisons de lisibilité, il a été considéré que mélanger le code "normal" et le traitement des erreurs était un style de programmation perfectible.

Gestion des exceptions

Syntaxe d'une exception

BEGIN
[programme susceptible de générera une erreur]
EXCEPTION WHEN
[erreur susceptible d'être générée]
THEN
[action à faire dans le cas d'erreur]
END;

Deux types d'exception :

- Interne
  - exception oracle pré-définie
  - exception oracle non pré-définie
- Externe (exception définie par l'utilisateur)

Exceptions prédéfinies

Bon nombre d'exceptions sont prédéfinies par Oracle, par exemple

- NO DATA FOUND est levée quand la requête d'une instruction de la forme SELECT ... INTO ... ne retourne aucune ligne.
  ATTENTION: cela ne veut pas dire que la valeur retournée est NULL.
- TOO MANY ROWS est levée quand plusieurs lignes répondent aux critères de sélection, mais une seule ligne peut être renvoyée.
- DUP VAL ON INDEX est levée si une insertion (ou une modification) est refusée à cause d'une contrainte d'unicité.

Gestion des exceptions

Exemple pour l'exception "No data found"

```
DECLARE
VAR_ENAME VARCHAR2(50);
BEGIN
VAR_ENAME := '';
SELECT ENAME INTO VAR_ENAME FROM EMP WHERE
EMPNO = 7788 AND 1 = 2;
DBMS_OUTPUT.PUT_LINE('VAR_ENAME = ' || VAR_ENAME);
END;
```

Rapport d'erreur :

ORA-01403: aucune donnée trouvée
ORA-06512: à ligne 6
01403. 00000 - "no data found"

Gestion des exceptions

Exemple pour l'exception "No data found"

```
DECLARE
    VAR_ENAME VARCHAR2(50);
BEGIN
    VAR_ENAME := '';
    SELECT ENAME INTO VAR_ENAME FROM EMP WHERE
EMPNO = 7788 AND 1 = 2;
    EXCEPTION WHEN NO_DATA_FOUND THEN
VAR_ENAME := 'Pas de donnée';
    END;
    DBMS_OUTPUT.PUT_LINE('VAR_ENAME = ' || VAR_ENAME);
END;
```

Le résultat maintenant est :

VAR_ENAME = Pas de donnée

Gestion des exceptions

```
DECLARE
num NUMBER;
nom VARCHAR2(30) := 'Poupée Batman ' ;
BEGIN
SELECT numprod INTO num
FROM PRODUIT
WHERE nomprod = nom ;
DBMS_OUTPUT . PUT_LINE ( 'L 'article' || nom || ' a pour numéro ' ||
num ) ;
EXCEPTION WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT . PUT_LINE ( 'Aucun article ne porte le nom || nom ) ;
WHEN TOO_MANY_ROWS THEN
DBMS_OUTPUT. PUT_LINE ( ' Plusieurs articles ont le même nom' ||
nom ) ;
WHEN OTHERS THEN
DBMS_OUTPUT. PUT_LINE ( ' Il ya un gros problème . . . ' ) ;
END;
/
```

Gestion des exceptions

Déclarer et lancer ses propres exceptions

Exception est un type, on déclare donc les exceptions dans une section DECLARE. Une exception se lance avec l'instruction RAISE.

```
DECLARE
Erreur_comm exception ;
v_pilot pilote%rowtype ;
BEGIN
Select * into v_pilot From PiloteWhere nopilot = '7100' ;
If v_pilot.comm > v.pilot.sal Then
Raise erreur_comm ;
.....
EXECPTION When erreur_comm then
Insert into erreur values(v_pilot.nom, 'Commission > salaire') ;
When NO_DATA_FOUND Then
Insert into erreur values(v_pilot.nopilot, 'non trouvé') ;
END ;
```

Plan

Procédures et fonctions

Procédure

- Les procédures stockées (PROCEDURE) sont des blocs PL/SQL nommés qui son stockés dans la base de donnée ORACLE avec un nom unique.
- Ces procédures comportent un programme qu'on peut l'appeler dans une autre procédure, d'une fonction, trigger ou d'un programme externe qui a accès à notre base de données.

Syntaxe

CREATE [OR REPLACE] PROCEDURE <NOM_PROCEDURE>
([VAR1 IN <TYPE_VAR1>],[VAR1 IN <TYPE_VAR1>], ...
[VARN OUT <TYPE_VARN>]) AS
[Déclaration variables à utiliser]
BEGIN
[Corps du programme]
END; /

Syntaxe d'une procédure

- CREATE : Créé la procédure stockée <NOM_PROCEDURE>
- OR REPLACE : Facultatif mais vaut mieux l'utiliser. Cela permet de recréer la PROCEDURE <NOM_PROCEDURE> si elle existe déjà.
- PROCEDURE : Mot clé pour définir qu'il s'agit d'une procédure stockée.
- <NOM_PROCEDURE> : Le nom de la PROCEDURE à créer. ([VAR1 IN <TYPE_VAR1>], .., [VARN OUT <TYPE_VARN>]) : Les paramètres de notre PROCEDURE, NOM_VARIABLE, IN ou OUT ou IN OUT et le type du paramètre (VARCHAR2, NUMBER, DATE, etc..).
  - IN : paramètre en entrée.
  - OUT : paramètre en sortie.
  - IN OUT : paramètre en entrée et en sortie. Les paramètres sont facultatifs. Nous pouvons déclarer une PROCEDURE sans paramètres d'entrées ou de sorties.

Syntaxe d'une procédure

- AS : Mot clé pour déclarer le début de la PROCEDURE. Variables à utiliser : Déclaration des variables qu'on va utiliser dans notre PROCEDURE.
- BEGIN : Début du bloc PL/SQL de la PROCEDURE.
- Corps du programme : Bloc PL/SQL de la PROCEDURE.
- END; : Fin de notre PROCEDURE.

PROCEDURE qui ompte le nombre d'employés pour un département donné

```
CREATE OR REPLACE PROCEDURE proc_dept (p_no IN
dept.deptno%TYPE)
IS
v_no NUMBER;
BEGIN
SELECT COUNT(deptno) INTO v_no FROM emp WHERE
deptno=p_no;
DBMS_OUTPUT.PUT_LINE('Nombre d'employés : '||' '|| v_no);
END;
/
```

Procédures et fonctions

Fonctions

- Les FUNCTION comme pour les PROCEDURE sont du bloc PL/SQL stockés dans l'a BD ORACLE.
- Les fonctions retournent toujours une valeur d'un type ORACLE définie.

Syntaxe d'une fonction

CREATE [OR REPLACE] FUNCTION
<NOM_FUNCTION>([VAR1 IN <TYPE_VAR1>],
[VAR1 IN <TYPE_VAR1>],
.... [VARN OUT <TYPE_VARN>])
RETURN <TYPE_RETOUR>
IS
[Déclaration variables à utiliser]
BEGIN
[Corps du programme]
END;

Syntaxe d'une fonction

- CREATE : Créé la fonction <NOM_FUNCTION>.
- OR REPLACE : Facultatif mais vaut mieux l'utiliser. Cela permet de recréer la FUNCTION
- <NOM_FUNCTION> si elle existe déjà.
- FUNCTION : Mot clé pour définir qu'il s'agit d'une fonction.
- <NOM_FUNCTION> : Le nom de la FUNCTION à créer.
- ([VAR1 IN <TYPE_VAR1>], .., [VARN OUT]) : Les paramètres de notre FUNCTION, NOM_VARIABLE, IN ou OUT ou IN OUT et le type du paramètre (VARCHAR2, NUMBER, DATE, etc..).

Syntaxe d'une fonction

- IS : Mot clé pour déclarer le début de la FUNCTION.
- Variables à utiliser : Déclaration des variables qu'on va utiliser dans notre FUNCTION.
- BEGIN : Début du bloc PL/SQL de la FUNCTION. Corps du programme : Bloc PL/SQL de la FUNCTION.
- END; : Fin de notre FUNCTION.
- / : Permet de créer la FUNCTION.

Fonction qui compte le nombre d'employés pour un département donné

```
CREATE OR REPLACE FUNCTION proc_dept (p_no IN
dept.deptno%TYPE)
RETURN NUMBER
AS
v_no NUMBER;
BEGIN
SELECT COUNT(deptno) INTO v_no FROM emp WHERE
deptno=p_no;
RETRUN (v_no);
END;
/
```

Procédures et fonctions

FUNCTION qui retourne la date Système sous forme de VARCHAR2 formater en DD/MM/YYYY HH:MI:SS

```
CREATE OR REPLACE FUNCTION DATE_DU_JOUR () RETURN
VARCHAR2
IS
BEGIN
RETURN TO_CHAR(SYSDATE, 'DD/MM/YYYY HH24:MI:SS');
END;
/
```

On fait appel à cette fonction dans une requête

```
SELECT DATE_DU_JOUR FROM DUAL;
```

Résultat

```
DATE_DU_JOUR
-------------------
03/01/2017 18:33:52
```