Chapitres du cours

- Chapitre 1 : Introduction aux BD et SGBD
- Chapitre 2 : Le langage SQL
- **Chapitre 3 : Bases du langage PL/SQL**

Chapitre 3

Le Langage PL/SQL

Plan du chapitre 2

- Principes et structure générale d'un programme
- Gestion des exceptions
- Sélection et manipulation de données
- Sous-programmes et paquetages

Plan du chapitre 2

- **Principes et structure générale d'un programme**
- Gestion des exceptions
- Sélection et manipulation de données
- Sous-programmes et paquetages

PL/SQL ?

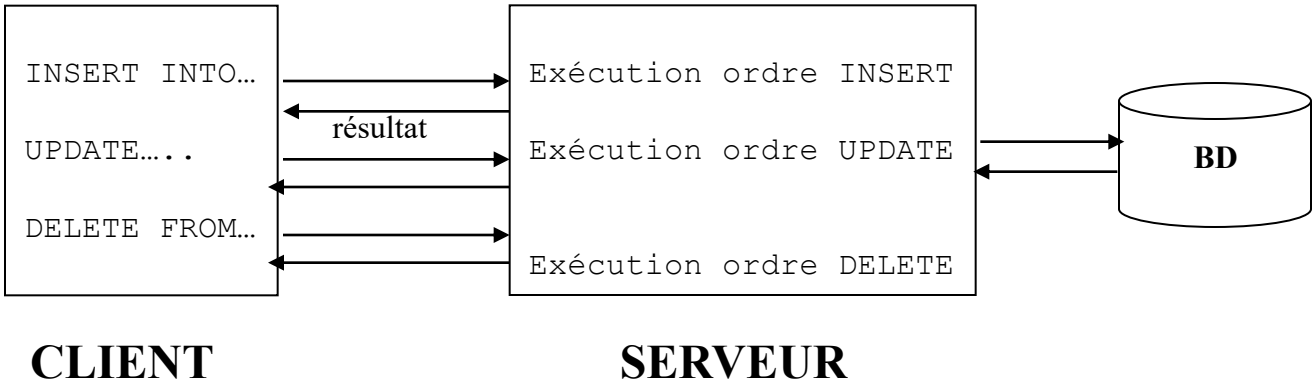
❑ Extension de SQL

- Procedure Language/Strctured Query Language
  - Langage procédural d'ORACLE depuis la version 6
  - Intègre des structures de contrôle (conditionnelle, choix multiples, boucle)
  - Manipule des données extraites par des instructions SQL

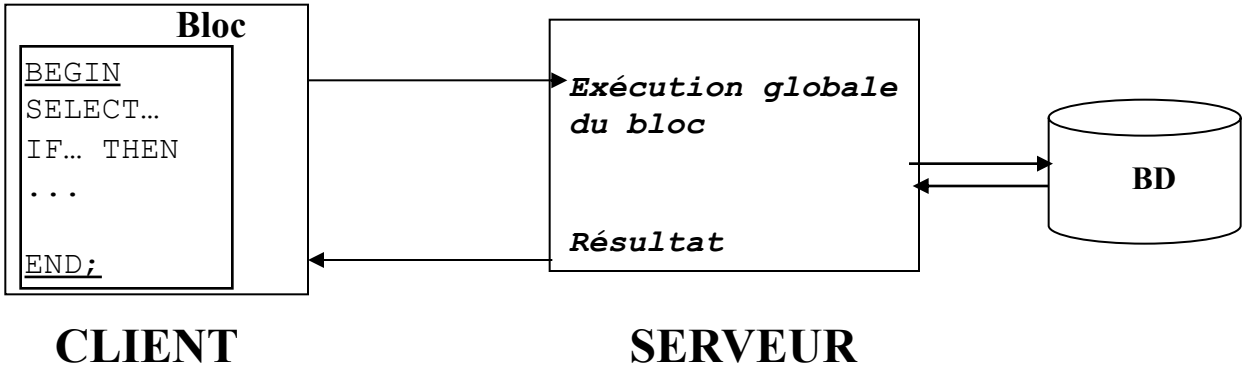
- Mode d'exploitation Client-Serveur
  - Instruction SQL => envoi de message client-serveur
  - Réponse/résultats serveur-client
  - Bloc PL/SQL = 1 message client-serveur

SQL Vs. PL/SQL ?

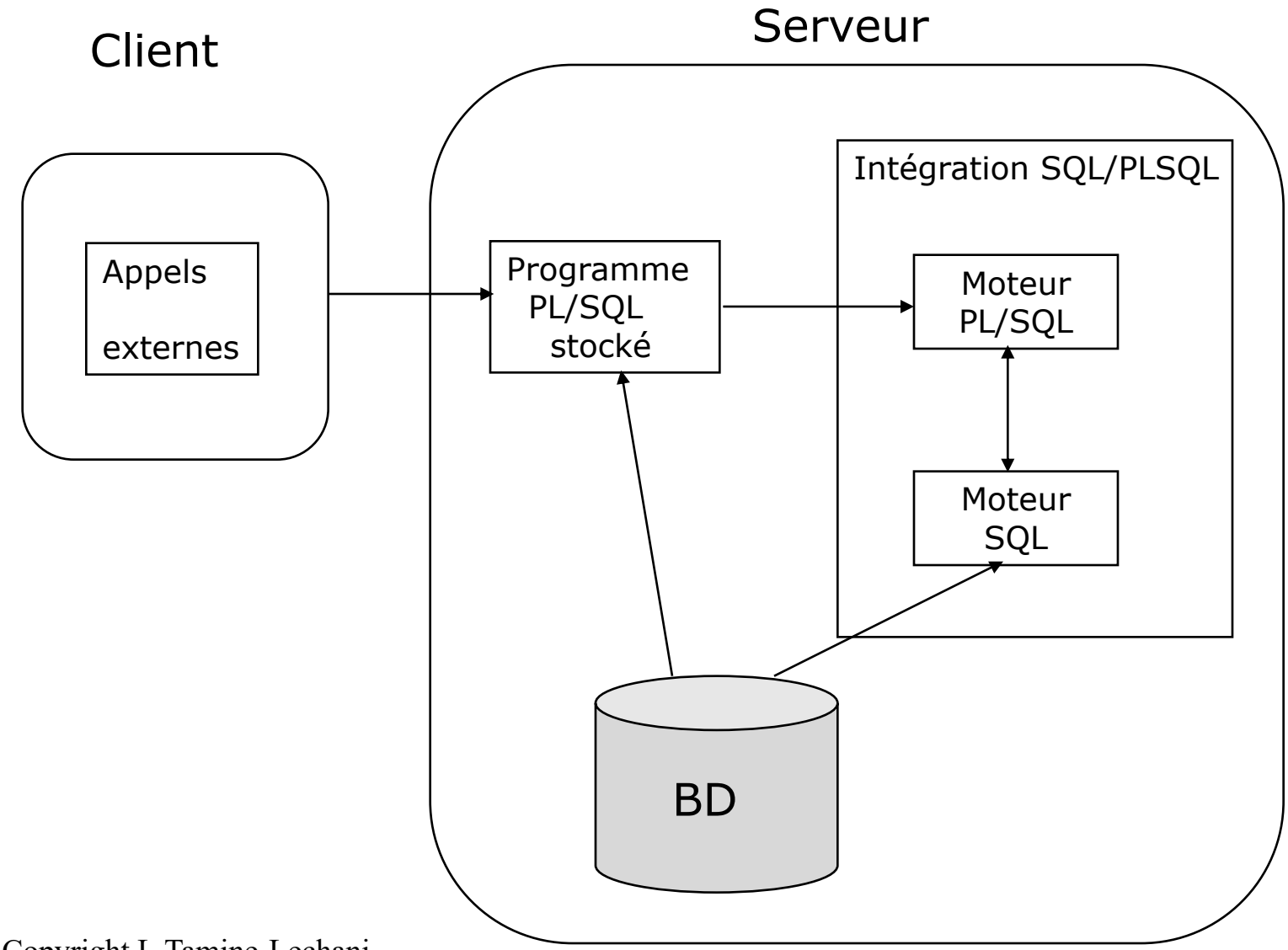
❑ SQL : Langage ensembliste, ordres SQL



❑ PL/SQL : Langage procédural, groupes d'ordres (blocs)



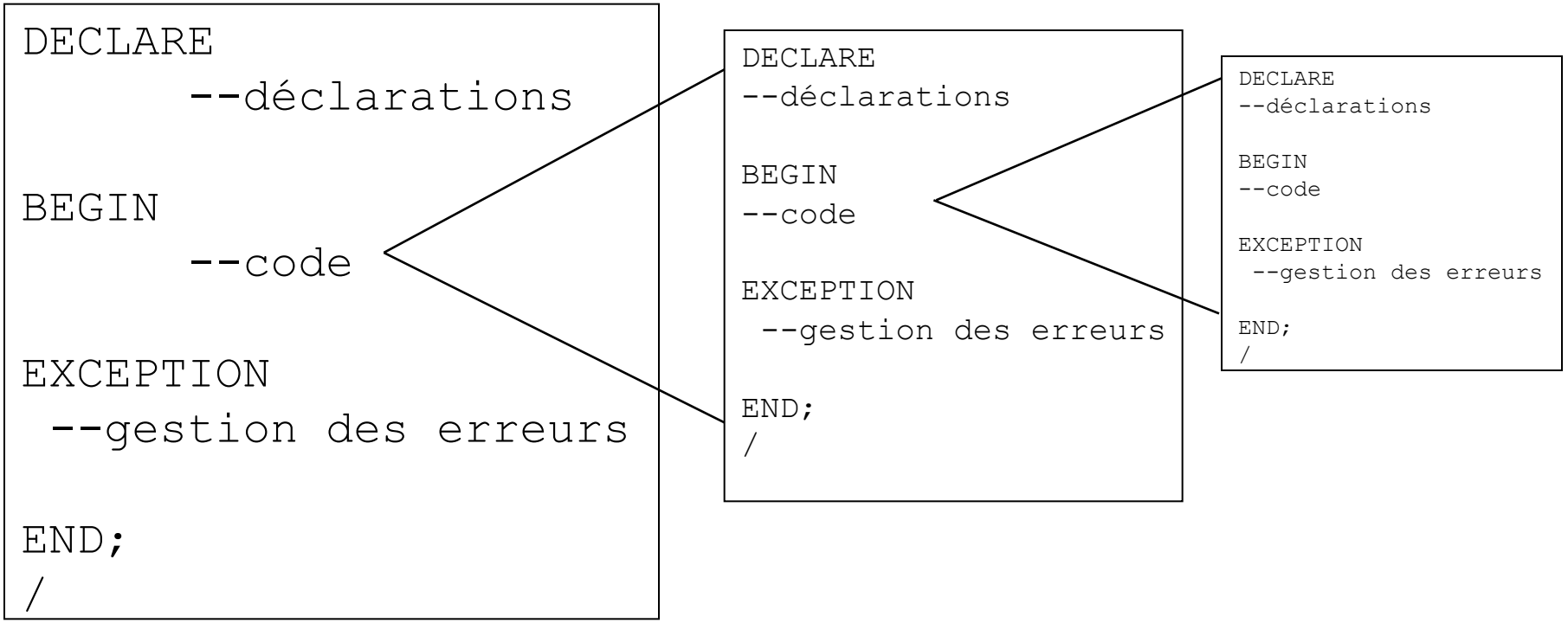
PL/SQL : fonctionnement sous Oracle



Structure d'un programme PL/SQL

- **Section DECLARE : section optionnelle**
  - Variables
  - Types
  - Curseurs
  - Exceptions
- **Section BEGIN : section obligatoire**
  - Ordres exécutables : SQL, PL
  - Fin signalée par « **END; /** »
- **Section EXCEPTION : section optionnelle**
  - Traitements d'erreurs
  - Exceptions SQL ou utilisateur

Structure d'un programme PL/SQL



Imbrication des blocs PL/SQL

Portée des variables

- **Visibilité des variables**
  - Un bloc a la visibilité des variables associées
  - Un bloc **a la visibilité** des variables des blocs **de niveaux inférieurs pendant la durée de vie de ces derniers**
  - Un bloc en cours d'évalualiation **a la visibilité** des variables des blocs **de niveaux supérieurs**

```
DECLARE
 a : Number
BEGIN
--code
--b non visible
--(en dehors de son bloc)
EXCEPTION
 --gestion des erreurs
END;
/
```

```
DECLARE
b : Number

BEGIN
--code   a, b visibles

EXCEPTION
 --gestion des erreurs

END;
/
```

Objets PL/SQL

Identificateurs	Même principe de nommage qu'en SQL
Commentaire	Monoligne : introduit par le symbole « -- » Multilignes : introduit par le Symbole « /* » et fermé par le symbole « */ »
Types de données	Domaines de valeurs des variables : prédéfinis, définis
Variables et constantes	Variable PL/SQL, variables non PL/SQL
Structures de contrôle	Conditionnelles, alternative, itératives

Types de données

❑ Types de données SQL + ….

VARCHAR2(size)	Données caractères de longueur variable
NUMBER(taille[,precision])	Numérique de longueur variable
DATE	Valeurs de date et d'heure
LONG	Données caractères de longueur variable (2Go)
CLOB	Données caractères (4 Go)
RAW	Binaire
BLOB	Binaire, jusqu'à 4 Go
BFILE	Binaire, stocké dans un fichier externe (4 Go)

Types de données

Types et sous-types spécifiques à PL/SQL

BINARY_INTEGER	Sous-type de NUMBER, entiers signés -2^{31} à 2^{31}
PLS_INTEGER	Sous-type de NUMBER, entiers signés -2^{31} à 2^{31} (≠ BINARY_INTEGER dans les performances)
NATURAL, POSITIVE	Sous type de Binary_Integer, non négatif
NATURALN, POSITIVEN	Sous type de Binary_Integer, non négatif, non null
SIGNTYPE	Domaine de valeurs {-1, 0, 1}
DEC, DECIMAL, NUMERIC	Décimaux, précision de 38 chiffres
DOUBLE PRECISION, FLOAT REAL	Flottants

Opérateurs

❑ Opérateurs arithmétiques standards

**	Exponentiation
*	Multiplication
/	Division
+	Addition
-	Soustraction
||	Concaténation
:=	Affectation

Opérateurs

❑ Opérateurs relationnels standards

=	Egal
<>, !=	Différent
>	Supérieur
<	Inférieur
>=	Supérieur ou égal
<=	Inférieur ou égal

Variables et constantes simples

❑ Variables et constantes simples

```
Identificateur [CONSTANT] TypeDeDonnées [NOT NULL]
[:=|DEFAULT expression];
```

Précise que c'est une constante

Précise la valeur par défaut

Pas de valeur nulle

Convention de nommage

```
vNb    NUMBER(2);
vPreNom VARCHAR2(30);
vAdmis BOOLEAN NOT NULL DEFAULT TRUE;
.............
```

Possible lorsque la variable est déjà déclarée

Variables référençantes

- **Variables %TYPE**
  - Déclare une variable selon
    - ✓ la définition d'une colonne, d'une table ou d'une vue existante
    - ✓ d'une autre variable

```
vSalaire     Employes.salaire%TYPE;
vSpec        Etudiant.specialite%TYPE;
.......
```

Variables référençantes

❑ Variables %ROWTYPE

- Déclare une variable selon la structure d'un tuple (ligne) : tous les attributs ou partie des attributs
- Chaque attribut de variable typée par %ROWTYPE est instanciable
- Chaque variable typée %ROWTYPE est manipulable en bloc

```
vEmploye     Employes%ROWTYPE;
vEtud        Etudiant%ROWTYPE;
.............................

.............
............
vEmploye.Prenom:='TITI';
.................................
```

Variables enregistrement

❑ Variables RECORD

- Equivalentes à %ROWTYPE pour des structures personnalisées
- Type RECORD à définir, peut être imbriqué

```
Type NomRecord IS RECORD
(nomChamp typeDonnées [[NOT NULL] {:=|DEFAULT}
expression][,nomChamp typeDonnees…]…);
```

```
Type Appartement IS RECORD
(Numero INTEGER NOT NULL,
 Immeuble CHAR NOT NULL,
 Sup NUMBER(3,2));

.........................
.........................
```

Variables tableaux

❑ Variables TABLE

- Tableaux à dimension non prédéfinie, dynamiques
- Composé : clé primaire, colonne

```
Type NomTypTableau IS TABLE OF
{typscalaire| variable%TYPE
|table.colonne%TYPE [NOT NULL] |table.%ROWTYPE
[INDEX BY BINARY_INTEGER]};

vnomTableau nomTypTableau;
```

Elément de type simple

Variable type tableau

Elément de type ROWTYPE

- Référence à un élément du tableau

```
vElementtableau : vnomTableau (rangdstableau);
```

Indice de l'élément dans le tableau

Variables tableaux : Exemple

- **Variables TABLE**

```
Type MarqueVoiture IS TABLE OF Voiture.Marque%TYPE
INDEX BY BINARY_INTEGER;
.....................................
INDEX BY BINARY_INTEGER;

vMarques MarqueVoiture; MesVoitures Voitures;
i Number;

BEGIN
vMarques(1):= 'PEUGEOT';
vMarques(2):= 'RENAULT';
…
.....................................
```

Fonctions pour les tableaux

EXISTS(x)	Retourne TRUE si le xième élément du tableau existe
COUNT	Retourne le nombre d'éléments du tableau
FIRST, LAST	Retourne le premier/dernier indice du tableau (NULL si tableau vide)
PRIOR(x), NEXT(x)	Retourne l'élément avant/après le xième élément du tableau
DELETE	Supprime un ou plusieurs éléments
DELETE (x)	Attention, ces fonctions NE PEUVENT PAS ETRE appelées dans une requête SQL (SELECT, INSERT, UPDATE OU DELETE)
DELETE (x,y)	

Variables de substitution

❑ Variables d'interface SQL - PL/SQL

- Variables passées en paramètres d'entrée bloc PL/SQL
- Variables préfixées par le symbole « & »
- Utilisation des commandes ACCEPT…PROMPT pour la saisie des variables paramètres

Variable non déclarée

```
ACCEPT param PROMPT 'Entrer la valeur :';
DECLARE
        -- déclarations
BEGIN
        -- code
        -- &param si numérique
        -- '&param' si chaîne de caractères
END;
/
-- Ordre SQL
```

Variables de substitution

❑ Variables d'interface SQL - PL/SQL

```
ACCEPT nometud PROMPT 'Nom Etudiant :';
ACCEPT prenometud PROMPT 'Prénom Etudiant :';
DECLARE
       vNom Etudiant.Nom%TYPE;
       vPren Etudiant.Prenom%TYPE;
       vDtn Etudiant.Dtn%TYPE;
BEGIN
      ..............................
      ......................
      ......................
      DBMS_OUTPUT.PUT_LINE ('Nom:'|| vNom || ' Prénom:'||
vPren || ' Date de naissance :'|| vPren);
END;
/
```

Variable type chaîne de caractères entre ''

Structures de contrôle

- **Structures génériques**
  - Structures conditionnelles
  - Structures alternatives
  - Structures itératives

- **Conditions, Expressions, Opérateurs**
  - Idem que SQL (voir cours SQL)

Structures de contrôle

❑ Structures conditionnelles

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

```
IF vTrouve THEN
DBMS_OUTPUT.PUT_LINE ('OK');
END IF;
```

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

```
IF (vIndice >4) THEN
 vSalaire:=vSalaire*1,4;
ELSE
vSalaire:=vSalaire*1,2;
END IF;
```

```
IF condition1 THEN
 instructions;
ELSIF condition 2 THEN
 instructions;
ELSE
 instructions;
END IF;
```

```
IF vNote>=16 THEN
  vMention:='TB';
ELSIF vNote>=14 THEN
 vMention:='B';
ELSIF vNote>=10 THEN
 vMention:='AB';
ELSE
  vMention:='R';
END IF;
```

Structures alternatives

❑ Structure alternative

```
[<<etiquette>>]
CASE variable
WHEN expr1 THEN instructions1;
WHEN expr2 THEN instructions2;
…
WHEN exprn THEN instructionsN;
[ELSE instructionsN+1;]
END CASE [etiquette];
```

```
CASE vSolde
WHEN vSolde >=100000
 THEN categorie:='A';
WHEN vSolde >=50000
 THEN categorie:='B';
WHEN vSolde >=1000
 THEN categorie:='C';
ELSE categorie:='D';
END CASE;
```

```
[<<etiquette>>]
CASE
WHEN val1 THEN instructions1;
WHEN val2 THEN instructions2;
…
WHEN valn THEN instructionsN;
[ELSE instructionsN+1;]
END CASE [etiquette];
```

```
CASE vNote
WHEN 'A'
 THEN mention:='TB';
WHEN 'B'
 THEN mention:='B';
WHEN 'C'
 THEN mention:='P';
END CASE;
```

Structures itératives

Boucles

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

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

```
FOR compteur IN [REVERSE]
valeurInf..valeurSup LOOP
 instructions;
END LOOP;
```

```
Vconverg:=FALSE;vGamma:=1;
WHILE not vconverg LOOP
 vAlpha:=(vAlpha-0,1)/vGamma;
 vGamma := vGamma*1,2;
IF vAlpha < vSeuil THEN
vConverg:=TRUE;
END IF;
END LOOP;
```

```
vSomme:=0;
FOR vCompt IN 2..19 LOOP
 vSomme:=vSomme+vCompt;
END LOOP;
```

Structures itératives

❑ Boucles avec étiquettes

```
<<Etiquette>>
LOOP

 instructions;

END LOOP <<Etiquette>>;
```

Permet des sorties « anticipées »
à partir de n'importe quel niveau de boucle

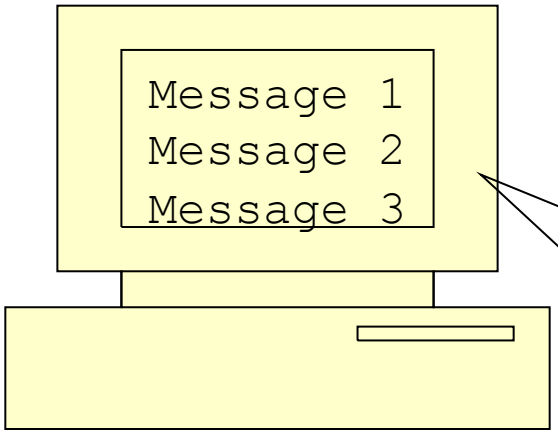
```
vTest:=0;
<<boucle1>>
LOOP
vVar1:=10;
LOOP
vVar2:=7;
LOOP
vVar3:=9;
EXIT boucle1 WHEN vVar3>100;
EXIT WHEN vVar3>20;
…
END LOOP;
….
END LOOP;
….
END LOOP;
```

Paquetage DBMS_OUTPUT : affichage des résultats

❑ Paquetage DBMS_OUTPUT

- Messages enregistrés dans une mémoire tampon côté serveur
- La mémoire tampon est affichée sur le poste client **à la fin**
- **Activation du paquetage : SET SERVEROUTPUT ON**

Client SQL*Plus




```
SQL> SET SERVEROUTPUT ON
```

Serveur Oracle

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Message 1');
DBMS_OUTPUT.PUT_LINE ('Message 2');
DBMS_OUTPUT.PUT_LINE ('Message 3');
END;
```

```
Message1
Message2
Message3
```

Paquetage DBMS_OUTPUT : affichage des résultats

❑ Principales procédures de DBMS_OUTPUT

PUT (ligne IN VARCHHAR2 | DATE NUMBER)	Mise dans le tampon d'un paramètre
NEW_LINE (ligne OUT VARCHAR2, statut OUT INTEGER)	Ecriture du caractère fin de ligne dans le tampon
PUT_LINE (ligne IN VARCHHAR2 | DATE NUMBER	PUT puis NEW_LINE
GET_LINE (ligne OUT VARCHAR2, statu OUT INTEGER)	Affectation d'une chaîne du tampon dans le tampon

Paquetage DBMS_OUTPUT : affichage des résultats

- **Appel des procédures de DBMS_OUTPUT**
  - DBMS_OUTPUT.nomProcedure (paramètres)

```
vSomme:=0;
.............................
 DBMS_OUTPUT.PUT_LINE ('Itération : '|| vCompt);
 ..........................
..............
DBMS_OUTPUT.PUT_LINE ('Somme = '|| vSomme);
```

Plan du chapitre 2

- Principes et structure générale d'un programme
- **Gestion des exceptions**
- Sélection et manipulation de données
- Sous-programmes et paquetages

BD exemple (voir Chapitre 1)

Produit (RefProd, Designation, Cat,Prix, Stock, PrixAch)

Commande (Refcomm, DateComm, IdCli#)

LigneCommande (Refcomm#, RefProd#, QtComm)

Client (IdCli, Nom, Prenom, Solde)

Mécanisme de gestion des exceptions

❑ Définition d'une exception

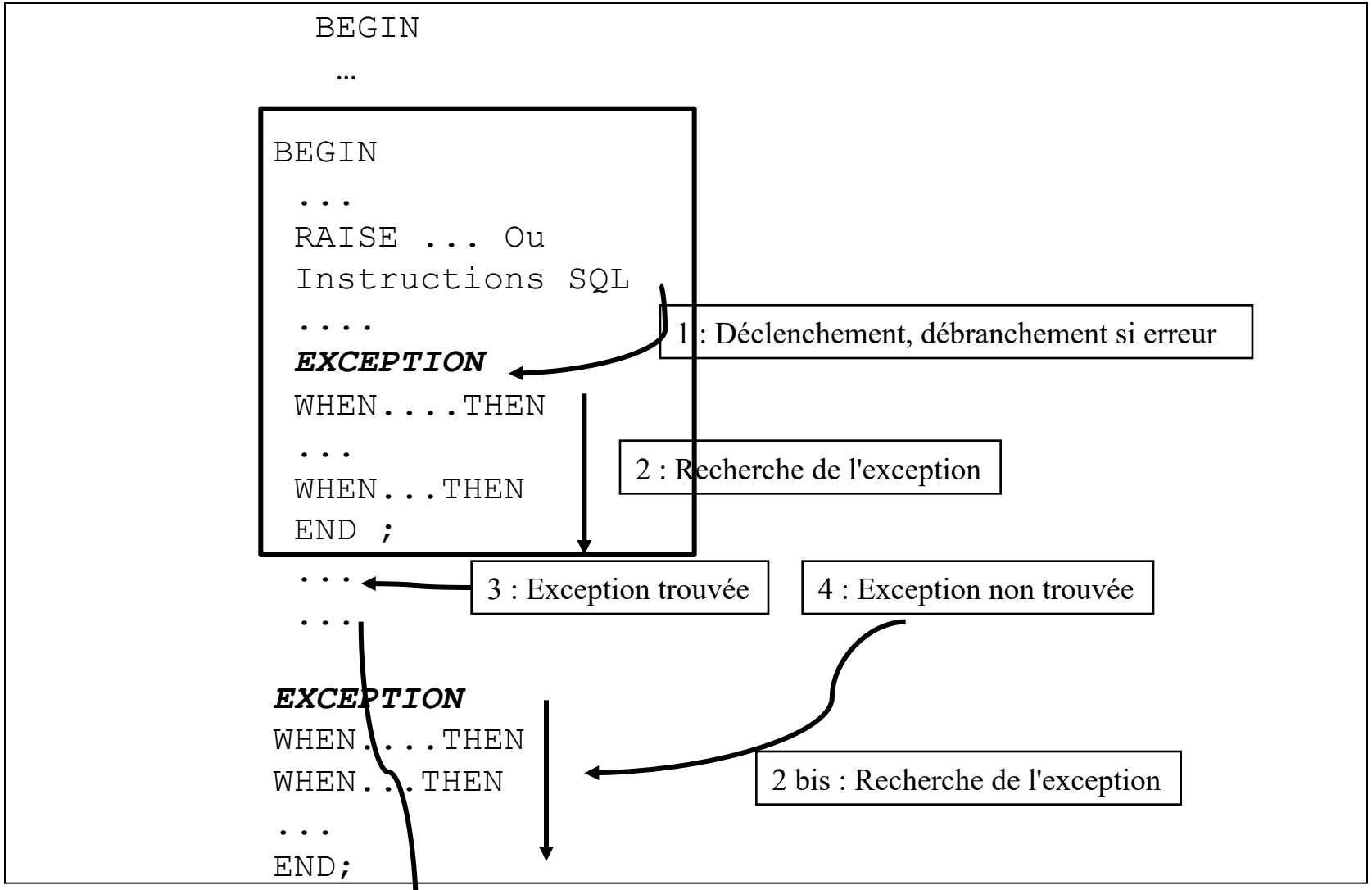
- ✓ Une exception est une erreur logicielle qui se produit à l'exécution d'un programme. Une exception non traitée est remontée via les appels de sous-programmes et cause l'arrêt du programme.

- ✓ A toute exception sont associés deux variables système consultatbles
  - o `sqlcode` : un code erreur prédéfinie (`ORA-XXXX`) ou personnalisé
  - o `sqlerrm` : le message d'erreur correspondant au `sqlcode`

❑ Une exception est :

- Levée (détectée) dans un bloc délimité par BEGIN …END
- Traitée dans un bloc EXCEPTION du programme

Principe de propagation des exceptions (1)



Principe de propagation des exceptions (2)

- Si aucune erreur ne se produit, le bloc EXCEPTION est ignoré et le traitement se termine normalement;
- Si une anomalie se produit, le bloc EXCEPTION est exécuté selon les règles suivantes

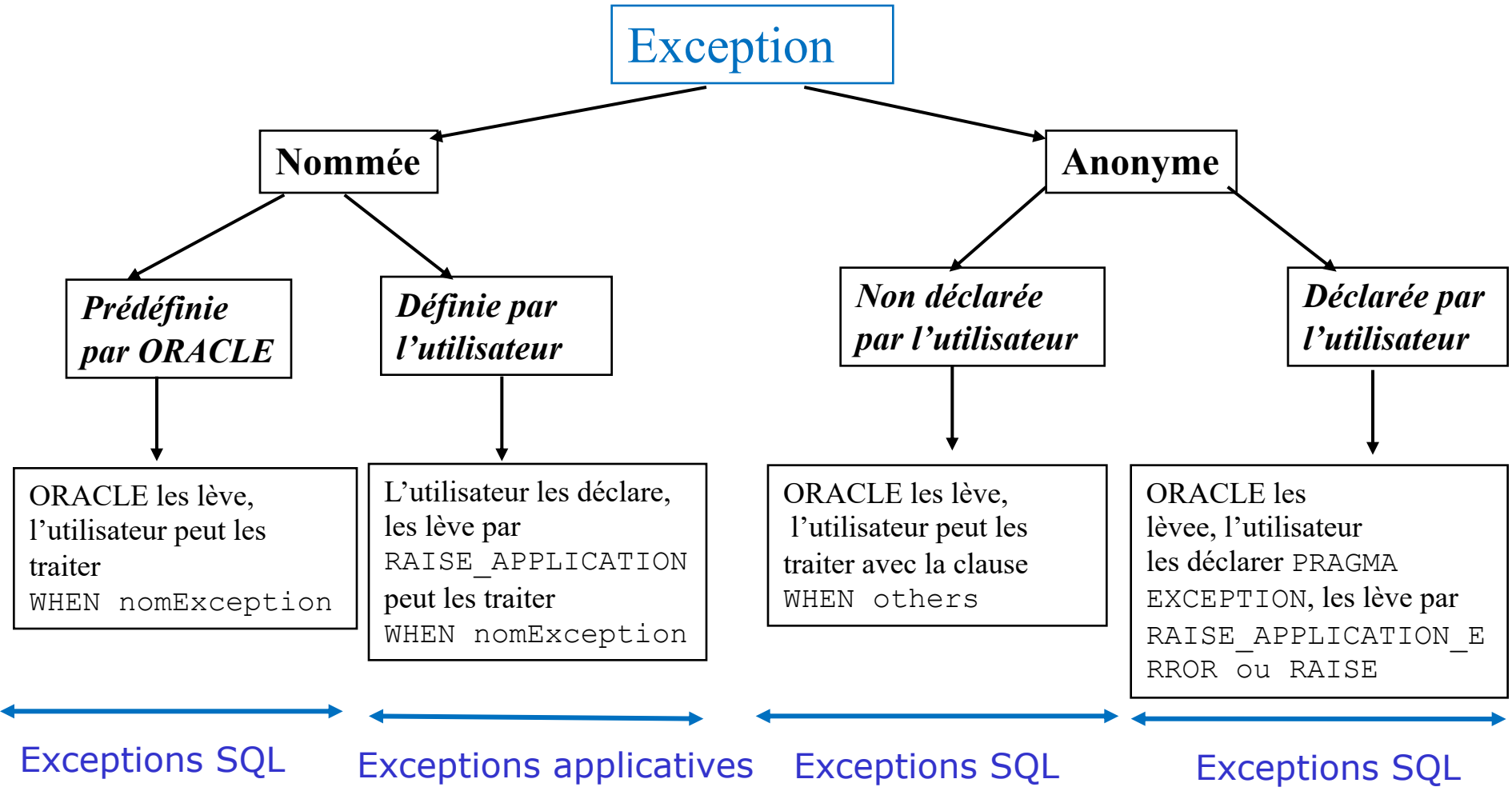
(a) Le type d'erreur est prévu est dans une des entrées WHEN, les instructions de cette entré sont exécutées et le programme se termine

(b) Le type d'erreur n'est pas parmi les entrées WHEN alors :

(*) s'il existe une entrée OTHERS, les instructions de cette entrée sont exécutées et le programme se termine

(*) s'il n'existe pas une entrée OTHERS, l'erreur est propagée au programme appelant

Typologie des exceptions



Quelques exceptions SQL (nommées ou anonymes)

CURSOR_ALREADY_OPEN	ORA-06511	Ouverture d'un curseur déjà ouvert
INVALID_CURSOR	ORA-01001	Ouverture interdite sur un curseur
NO_DATA_FOUND	ORA-01403	Requête SELECT ne retournant aucun résultat. Non opérationnel pour UPDATE, DELETE
ROWTYPE_MISTMATCH	ORA-06504	Incompatibilité de types entre variable externe et variable PL/SQL
TOO_MANY_ROWS	ORA-01422	Requête retournant plusieurs lignes
Exceptions ORACLE anonymes	ORA-01400	Tentative d'insertion d'une valeur null non autorisée
	ORA-01407	Tentative de mise d'une valeur null non autorisée
	ORA-02290	Violation d'une contrainte CHECK
	ORA-02290	Tentative d'insertion d'une valeur
	ORA-02291	FK inexistante
	ORA-02292	Tentative de suppression d'une valeur FK référencée

Quelques exceptions SQL

DUP_VAL_ON_INDEX	ORA-0001	Insertion d'une valeur de clé primaire déjà existante
ZERO_DIVIDE	ORA-1476	Erreur de division par zéro
NO_LOGGED_FOUND	ORA-1012	Accès sans connexion préalable
PROGRAM_ERROR	ORA-06501	Problème rencontré dans le programme
LOGIN_DENIED	ORA-01017	Nom reconnaissance du login, mot de passe

Traitement des exceptions

❑ Exceptions SQL

- Nommées (`DUP_VAL_ON_INDEX, NO_DATA_FOUND,...`)
  - Pas de déclaration
  - Traitement dans la section EXCEPTION

```
WHEN <ExceptionPredef> THEN……
```

- Anonymes
  - Déclaration obligatoire avec le numéro d'erreur (sqlcode)

```
nomException EXCEPTION;
PRAGMA EXCEPTION_INIT(nomException, numerreur);
```

  - Traitement dans la section EXCEPTION

```
WHEN nomException THEN……
```

Traitement des exceptions applicatives

❑ Exceptions applicatives

- Déclaration sans numéro d'erreur

```
nomException EXCEPTION;
```

- Levée explicite par l'utilisateur

```
RAISE nomException; OU
RAISE_APPLICATION_ERROR (coderr, 'message');
```

- Traitement dans la section EXCEPTION

```
WHEN nomException THEN……
```

Exceptions : Exemple

```
ACCEPT vcode PROMPT 'Entrer le code du produit : ';
DECLARE
vRefProd NUMBER; vPrix NUMBER; AvgPrix NUMBER; Nprix NUMBER; vPrAch
NUMBER; vnom varchar(30);
.................................
BEGIN
.........................................
.....................
.....................
UPDATE Produit SET Prix = Prix* 0.8 WHERE Refoprod=&Vcode;
        Nprix:=vPrix*0.8;
.........................................
DBMS_OUTPUT.PUT_LINE('Ancien prix : '|| vPrix ||'    Nouveau
prix:'||Nprix);

.....................
.....................
.............................
.............................
 .............................

END;
/
```

Exceptions : Exemple

```
ACCEPT vCode PROMPT 'Entrer le code du produit : ';
DECLARE
vCode NUMBER;
.....................................
.....................................
BEGIN
DELETE FROM Produit WHERE code=&vCode;

.....................
.....................
.....................................

END;
/
```

Exception pour absence de clé étrangère, anonyme, connue avec le sql code -2292

Déclenchement possible de l'exception si le produit qu'on veut supprimé est référencé dans une commande

Traitement de l'exception

Traitement des exceptions : Cas d'une même exception levée à différents point du bloc PL/SQL

```
DECLARE
Erreur EXCEPTION;
flag NUMBER;
BEGIN
       flag:=1;
       SELECT …
       flag:=2;
       SELECT…
       IF … THEN RAISE erreur;
EXCEPTION
       WHEN NO_DATA_FOUND THEN
              IF flag=1 THEN …
                     ELSE …
              END IF;
       WHEN Erreur THEN ….
       WHEN OTHERS THEN …
END;
 /
```

Exception prédéfinie : pas de données trouvées, peut être générée par deux instructions différentes, différencier par une variable (flag)

Exception définie par l'utilisateur

Toute autre erreur

Sélection de données

❑ Principes de sélection des données

- Extraction mono-ligne, affectation de valeurs
  - `SELECT ...INTO varRowtype....FROM`
  - `SELECT A, B, ..., Z INTO var1, var2, ...varn`

- Extraction multi-lignes
  - Boucle sur un résultat

Sélection de données

❑ Sélection mono-ligne

- Principe du SELECT vu en SQL : **renvoi d'1 enregistrement**

```
SELECT liste INTO nomvarPLSQL [,nomvarPLSQL]
FROM <table>
[WHERE <condition>]
[GROUP BY <colonne>|<expression>]
[HAVING <condition>]
[ORDER BY <colonne>|<expression> [asc|desc]];
```

Précise les variables contenant les valeurs renvoyées.
Ces variables doivent être déclarées

Erreurs possibles : TOO_MANY_ROWS, NO_DATA_FOUND

Sélection de données : Exemple

```
ACCEPT vRefProd PROMPT 'Entrer la référence du produit:';
DECLARE
........ Produit.Prix%TYPE;
......... Produit.Designation%TYPE;

BEGIN
.........................
.............
.............
DBMS_OUTPUT.PUT_LINE('La désignation et prix du
produit sont :'||vDesign ||vPrix);
```

Sélection de données : *erreurs possibles*

```
SELECT QtiComm INTO vQte
FROM LigneCommande
WHERE RefComm='ZR456';
```

Erreur ORA-01422:
l'extraction exacte ramène plus que le nombre de lignes demandées

```
SELECT IdCli INTO vIdCli
FROM COMMANDE
WHERE RefComm='****';
```

Erreur ORA-01403:
aucune donnée trouvée

Sélection de données

❑ Sélection multi-lignes : notion de CURSEUR

- Obligatoire pour sélectionner plusieurs lignes
- Zone mémoire associée à une ligne renvoyée par un **SELECT**
- Curseur : adresse de la ligne en cours de traitement

RefProd	Designation	Prix	Stock
18	Table rotative	300	68
12	Chaise longue	57	50
15	Bureau	450	12
23	Armoire	230	1

Curseur

Sélection de données

❑ Traitement de CURSEUR

- **Manuel** : traitement en quatre (4) étapes
  - Déclaration dans la section **DECLARE**
  - Ouverture **OPEN**
  - Chargement et parcours **FETCH**
  - Fermeture : **CLOSE**

- **Semi-automatique** : traitement en deux (2) étapes
  - Déclaration dans la section **DECLARE**
  - Chargement et parcours **FETCH**

- **Automatique (CURSEUR temporaire)**
  - Chargement et parcours **FETCH**

Sélection de données

Commandes pour les curseurs

`CURSOR nomCurseur IS requête`	Déclaration du cursor
`OPEN nomCurseur`	Ouverture du curseur (chargement des lignes). Aucune exception n'est levée si la requête ne ramène aucune ligne
`FETCH nomCurseur INTO listeVariables |nomRecord`	Positionnement sur la ligne suivante et chargement de l'enregistrement courant dans une ou plusieurs variables
`CLOSE nomCurseurC`	Ferme le curseur. L'exception INVALID_CURSOR se déclenche si des accès au curseur sont opérés après sa fermeture

Sélection de données

Commandes pour les curseurs

nomCurseur%ISOPEN	Retourne TRUE si le curseur est ouvert, FALSE sinon
nomCurseur%NOTFOUND	Retourne TRUE si le dernier FETCH n'a pas renvoyé de ligne (fin de curseur
nomCurseur%FOUND	Retourne TRUE si le dernier FETCH a retourné une ligne
nomCurseur%ROWCOUNT	Retourne le nombre total de lignes traitées jusqu'à présent

Traitement manuel de CURSEUR

```
DECLARE                                          1. Déclaration
.........................................
vDesign Produit.Designation%TYPE;
vPrix   Produit.Prix%TYPE;
p1,p2 NUMBER;
BEGIN                          2. Ouverture
        ..............
        ..................                3. Lecture
       p1:=0; p2:=0;
        WHILE c1%FOUND LOOP
                IF vPrix > 1000 THEN p1:=p1+1;
                        INSERT INTO prix_eleve VALUES (vDesign);
                ELSE p2:=p2+1;
                        INSERT INTO prix_abord VALUES (vDesign);
                END IF;
                ..........................
        END LOOP;                  4. Fermeture
        ..........
        DBMS_OUTPUT.PUT_LINE(p2 ||' produits à prix abordables');
        DBMS_OUTPUT.PUT_LINE(p1 ||' produits chers');
        COMMIT;
END;
/
```

Traitement manuel de CURSEUR

```
DECLARE
.....................................
.................
vDesign Produit.Design%TYPE;
vPrix   Produit.Prix%TYPE;
p1,p2 NUMBER;
BEGIN
        ............
        ..............
         p1:=0; p2:=0;
        WHILE c1%FOUND LOOP
                IF vPrix > 1000 THEN p1:=p1+1;
                        INSERT INTO prix_eleve VALUES (vDesign);
                ELSE p2:=p2+1;
                        INSERT INTO prix_abord VALUES (vDesign);
                END IF;
                ...........................
        END LOOP;
        ..........
        DBMS_OUTPUT.PUT_LINE(p2 ||' produits à prix abordable');
        DBMS_OUTPUT.PUT_LINE(p1 ||' produits chers');
        COMMIT;
END;/
```

Variable structure

2. Ouverture

3. Lecture

4. Fermeture

Traitement semi-automatique de CURSEUR

```
DECLARE
CURSOR c1 IS SELECT nom, prix FROM Produit ORDER by 1;
-- pas de déclaration de variables de réception
p1,p2 NUMBER;
BEGIN
        -- pas d'ouverture du curseur
        -- pas de fetch
       p1:=0; p2:=0;
        ......................
        IF c1_ligne.prix > 1000 THEN p1:=p1+1;
                INSERT INTO produit_chers VALUES (c1_ligne.Design);
        ELSE p2:=p2+1;
                INSERT INTO produit_abord VALUES (c1_ligne.Design);
                END IF;
        ..................
        -- pas de close
        DBMS_OUTPUT.PUT_LINE(p2 ||' produits à prix abordable');
        DBMS_OUTPUT.PUT_LINE(p1 ||' produits chers');
        COMMIT;
END;
 /
```

Variable STRUCT de réception

Traitement semi-automatique de CURSEUR

```
DECLARE
.....................................
.....................................
.....................................
.....................................
.....................................
BEGIN
        DBMS_OUTPUT.PUT_LINE('Les produits les plus vendus
                                        depuis le 01-01-2012');
        ......................
                DBMS_OUTPUT.PUT_LINE(c_ligne.Code ||' : '
                                || c_ligne.TotalCommande);
        END LOOP;
END;
 /
```

Traitement automatique de CURSEUR

```
DECLARE
-- pas de déclaration du curseur
p1,p2 NUMBER;
BEGIN
       p1:=0; p2:=0;
        ..........................................
                        ..........................
              IF c1_ligne.prix > 1000 THEN p1:=p1+1;
                     INSERT INTO produits_chers VALUES (c1_ligne.nom);
              ELSE p2:=p2+1;
                     INSERT INTO produits_abordables
                      VALUES (c1_ligne.nom);
              END IF;
        END LOOP;
        DBMS_OUTPUT.PUT_LINE(p2 ||' produits abordables');
        DBMS_OUTPUT.PUT_LINE(p1 ||' produits chers');
        COMMIT;
END;
 /
```

Requête du curseur

Traitement automatique de CURSEUR, CURSEUR paramétré

❑ Objectif : possibilité de réutilisation d'un curseur

- Différents curseurs (logiques) pour différents paramètres

- Passage de paramètres
  - ✓ à l'ouverture (OPEN) dans le cas de curseurs manuels
  - ✓ en cours de traitement de la boucle (FOR) dans le cas de curseurs semi-automatique, automatique

Traitement automatique de CURSEUR, CURSEUR paramétré

```
-- Bloc permettant de sélectionner le nom des produits correspondant
-- à une catégorie passée en paramètre et ayant un prix
-- supérieur à une valeur également passée en paramètre
DECLARE
.....................................
-- nom et type des paramètres sans contraindre la longueur
        .............................
        .............................
BEGIN
        OPEN Produits('A',150);
        …
END;
/
```

Paramètres du curseur

```
BEGIN
        ................................
                -- utilisation de ligne.nomEmp
        ..........
END;
/
```

CURSEUR dynamique

❑ Type REF CURSOR

- Permet de définir un **curseur dynamique**, sans le décrire au préalable par une requête

- Types de curseurs REF CURSOR
  - ✓ Curseur **typé** : précise le type de retour
  - ✓ Curseur **non typé** : ne précise pas le type de retour, peut être associé à toute requête

- Traitement : équivalent à un curseur manuel
  - ✓ Ouverture : **OPEN FOR requête**
  - ✓ Lecture : **FETCH**
  - ✓ Fermeture : **CLOSE**

CURSEUR dynamique non typé

```
DECLARE
.....................
.....................
.....................
.....................
p1,p2 NUMBER;
BEGIN
        ..............................
                        ............
        ..............................
        ..............................
         IF vPrix > 10000 THEN p1:=p1+1;
                        INSERT INTO produits_chers VALUES (&vDesign);
                ELSE p2:=p2+1;
                        INSERT INTO produits_abordables VALUES (&vDesign);
                END IF;
       ..............................
        END LOOP;
       ..............................
        DBMS_OUTPUT.PUT_LINE(p2 ||' produits abordables');
        DBMS_OUTPUT.PUT_LINE(p1 ||' produits chers');
        COMMIT;
END;
```

Déclaration du type du Curseur non typé

Déclaration de la variable curseur

Ouverture du curseur

Chargement et parcours du curseur dynamique

CURSEUR dynamique typé

Déclaration du type du curseur typé

```
DECLARE
.....................................
.............
.............
```

Déclaration de la variable curseur

```
p1,p2 NUMBER;
BEGIN
        ..............................
                      ..............
        ......................
        ......................
                    INSERT INTO produits_chers VALUES (&vNom);
                     ..................
        END LOOP;
       ............
        COMMIT;
END;
 /
```

Ouverture du curseur

Chargement et parcours du curseur dynamique

Manipulation de données

❑ Mêmes principes de manipulation que SQL

- INSERT

```
INSERT INTO <nom_table> [(liste de colonnes)]
VALUES (liste des valeurs/
variables de substitution);
```

- UPDATE

```
UPDATE <nom_table>
SET colonne1 = valeur1/var1
   [, colonne2 = valeur2/var2]
[WHERE predicat];
```

- DELETE

```
DELETE FROM <nom_table>
[WHERE predicat];
```

Manipulation de données : E*xemple*

```
ACCEPT vCode PROMPT 'Entrer le code du produit  : ';
DECLARE
vRefProd NUMBER; vPrix NUMBER; vact VARCHAR; AvgPrix NUMBER; NPrix
NUMBER;

BEGIN
SELECT RefProd, Design, Prix, Cat INTO vRefProd, vDesign, vPrix, vCat
FROM Produit
WHERE RefProd=&vRefProd;
SELECT AVG(Prix) INTO AvgPrix
FROM Produit
WHERE Cat=vCat;

DBMS_OUTPUT.PUT_LINE('Le prix moyen des produits de la catégorie' ||
vCat est :'|| AvgPrix);

IF vPrix<=AvgPrix/2 THEN
.............................................
.................
END IF;
DBMS_OUTPUT.PUT_LINE('Ancien prix : '|| vPrix ||'    Nouveau
prix:'||NPrix);
END;
/
```

Plan du chapitre 2

- Principes et structure générale d'un programme
- Sélection et manipulation de données
- Gestion des exceptions
- **Sous-programmes et paquetages**

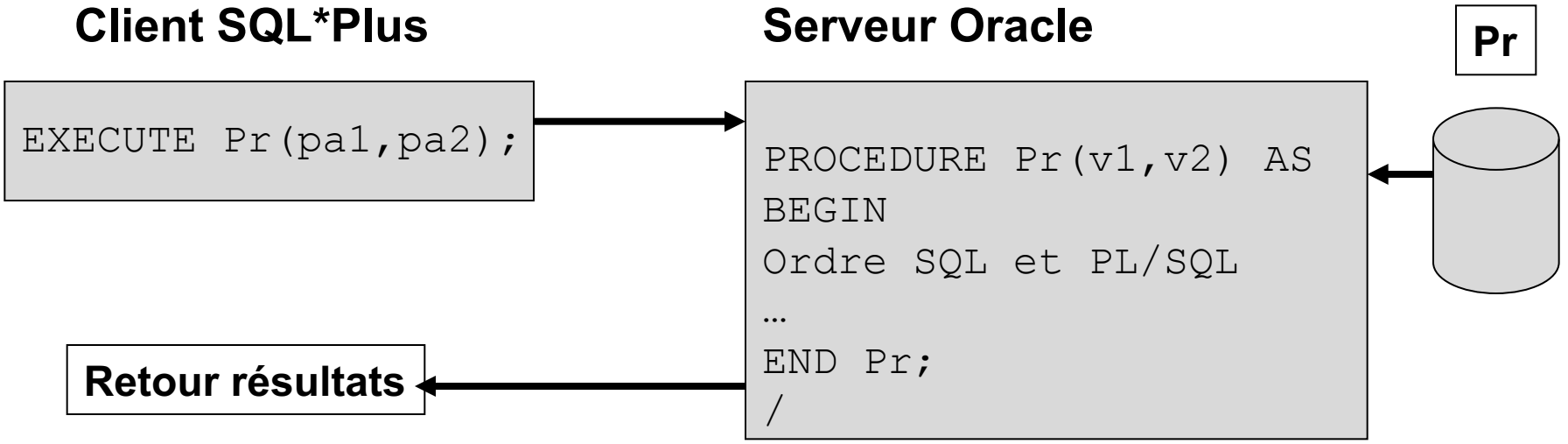
Sous-programme : c'est quoi ?

- Blocs PL/SQL nommés, avec paramètres en **entrée** et/ou **sortie**
- Deux types de sous-programmes
  - **Fonction** retourne un résultat unique
  - **Procédure** stockée retourne un ou plusieurs résultats
- Sous-programmes compilés et stockés dans la BD
  - Code source valide stocké dans USER_SOURCE
  - Code compilé optimisé : INDEX, CLUSTER, PARTITION, etc.
  - Code recompilé
    - automatiquement lors des modifications directs des objets
    - manuellement

```
ALTER PROCEDURE <nom_procedure> COMPILE;
```

- **Sous-programmes partagés** par plusieurs utilisateurs

Sous-programme : pourquoi ?



Sous-programme : pourquoi ?

- *Sécurité*
  - Droit d'accès sur les sous-programmes et non sur les objets du bloc associé (*GRANT EXECUTE ON NomProcedure to Util*)

- *Performance*
  - Programme compilé et optimisé
  - Réduction du nombre d'appels à la base
  - Partage de code

- *Productivité*
  - Simplicité de la maintenance des programmes : modularité, lisibilité, réutilisabilité, etc.

Procédures cataloguées : structure

```
CREATE [OR REPLACE] PROCEDURE <nom_procedure>
[(paramètre1 [ IN |OUT | IN OUT ] TypeSQL
     [{:=|DEFAULT} expression]
[, paramètre2 [IN |OUT | IN OUT ] TypeSQL
     [{:=|DEFAULT} expression]

…)]] {IS|AS}
-- déclarations des variables et
-- curseurs utilisés dans le corps de la procédure
BEGIN
…
-- instructions SQL ou PL/SQL
EXCEPTION
…
END;
/
```

Procédures cataloguées : exemple

```
CREATE or REPLACE PROCEDURE Inserer_LignComm(-------------------------,
-------------------------------------------------------------,
) ---------
Qte_Insuffisante EXCEPTION; vStock Produit.Stock%TYPE;
BEGIN
SELECT vStock FROM Produit WHERE RefProd=vvRefProd;
IF (vStock < vQte) THEN
RAISE Qte-Insuffisante
ELSE
UPDATE PRODUIT
SET Stock=Stock-vQte
WHERE RefProd=vRefProd;
INSERT INTO LigneComm VALUES (vref,vRefProd,vQte);
DBMS_OUTPUT.PUT_LINE('Insertion effectuee');
COMMIT;
END IF;
EXCEPTION
        WHEN Qte_Insuffisante THEN....
END;
/
```

Procédures cataloguées : exemple

```
CREATE PROCEDURE Supprimer_Prod (----------------------,
         --------------------) ----------
PRAGMA_EXCEPTION_INIT(Cde_En_Cours,-2292);
vRefProd Produit.RefProd%TYPE;
BEGIN
DELETE FROM Produit WHERE RefProd=vrefProd;
DBMS_OUTPUT.PUT_LINE('Produit '|| vRefProd ||' supprimé');
COMMIT;
Retour :=0;
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('Produit '|| vRefProd ||' inconnu');
-------------;
WHEN Cde_En_Cours THEN
DBMS_OUTPUT.PUT_LINE('Erreur : des commandes en cours pour ce produit');
------------;
…
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE(SQLCODE||'    '|| SQLERRM);
----------;
END;
/
```

Avec retour de valeur

Affectation de la valeur de sortie

Appel de procédure

- Sous SQL/Plus

```
EXECUTE <nomPROCEDURE> (list_param);
```

```
--exemple  SQL PLUS
set serveroutput on
EXECUTE Inserer_LignComm('F45','Estrade', 'C',50,100, 50);
```

```
--exemple  SQL PLUS
VARIABLE ret NUMBER
……
EXECUTE Supprimer_Prod(123, :ret);
IF ret=9 THEN…….
```

Appel de procédure

- Dans un programme PL/SQL

```
<nomPROCEDURE> (list_param);
```

```
ACCEPT vRefProd PROMPT 'entrez la référence du produit';
ACCEPT vDesign PROMPT 'entrez sa désigantion';
ACCEPT vCat PROMPT 'entrez sa catégorie';
ACCEPT vPrix PROMPT 'entrez son prix';
ACCEPT vStock PROMPT 'entrez son stock';
--Le prix à l'achat est le premier prix connu pour le produit
Ret NUMBER;
BEGIN
Inserer_Produit(&vRefProd, '&vDesign',&vPrix, &vCat,&vPrix,&vStock,
&vPrix);


……
END;
/
```

Fonctions

- *Principes*
  - Structure générale analogue à celle d'une procédure mais **retourne un résultat unique**
  - Retour de résultat avec la **clause RETURN**
  - Appels à partir :
    - d'une requête SQL
    - d'une procédure, fonction, programme PL/SQL
    - d'un programme externe

Fonctions : structure

```
CREATE [OR REPLACE] FUNCTION <nom_fonction>
[(variable1 type1,…,variablen typen )]
RETURN type_resultat IS

-- déclarations des variables, curseurs, exceptions
BEGIN
-- instructions SQL ou PL/SQL

RETURN(variable);
EXCEPTION
…
END;
/
```

Résultat à retourner typé

Retour du résultat

Fonctions : exemple

```
CREATE OR REPLACE FUNCTION Prix_Moyen
(------------------------------ ----)-----------
-------------------------------------
AvgPrix NUMBER(4,2);

BEGIN
SELECT AVG(Prix) INTO AvgPrix
FROM Produit
WHERE Categorie=vcat;

---------(AvgPrix);
END;
/
```

Appel de fonction

- A partir d'une requête SQL

Produits qui ne sont pas de la catégorie 'A' ayant un prix supérieur au prix moyen des produits de la catégorie 'A'

```
SELECT RefProd, Designation FROM Produit
WHERE Cat != 'A' AND Prix > PrixMoyen ('A');
```

- A partir d'une procédure ou fonction

```
BEGIN
….
CASE Prix_Moyen('A')
WHEN Prix_Moyen('B') >= 5000 THEN…
….
```

Sous-programmes récursifs

- Même principe que dans les autres langages

```
CREATE FUNCTION Factorielle (------------------)
------------------------

BEGIN
IF n=1 THEN RETURN 1;
ELSE
----------------------;
END IF;

END Factorielle;
/
```

Appel récursif

Sous-programmes imbriqués

- Principes

  - Décrit dans la partie déclarative d'un autre sous-programme

  - Durée de vie limitée au temps d'exécution du sous-programme qui l'imbrique

  - Dernier élément déclaré dans la partie déclarative du sous-programme qui l'imbrique

Sous-programmes imbriqués : structure

```
CREATE [OR REPLACE] PROCEDURE <nom_procedure>
[(paramètre1 [ IN |OUT | IN OUT ] TypeSQL
     [{:=|DEFAULT} expression]
[, ……

…)]] IS
-- déclarations des variables et
-- curseurs utilisés dans le corps de la procédure
PROCEDURE <nom_procedure_imb> [Parametre…..] IS
BEGIN
……

END <nom_procedure_imb>;
BEGIN
…
 Execute nom_procedure_imb (…);
EXCEPTION
…
END;
```

Déclaration de la procédure imbriquée

Appel de la procédure imbriquée

Sous-programmes imbriqués : exemple

```
CREATE OR REPLACE FUNCTION Puissance (a INTEGER, b INTEGER)
RETURN INTEGER IS
Ps INTEGER:=1; i INTEGER
-------------------------------------
FUNCTION Produit (n1 INTEGER, n2 INTEGER)
RETURN INTEGER IS
P INTEGER:=1;
BEGIN
P:=n1*n2;
RETURN P;
END Produit;
-------------------------------------
BEGIN
…
FOR i IN 1..b-1 LOOP
Ps:=Produit(Ps,a);
END LOOP;
END Puissance;
```

Définition de la fonction imbriquée

Appel de la fonction imbriquée

Paquetage

- *C'est quoi ?*
  - Ensemble de différents objets (variables, exceptions, sous-programmes etc.) formant un ensemble homogène

- *Structure*
  - Spécification ou **partie visible**
    - Comprend les signatures des sous-programmes, la déclaration des variables, curseurs, exceptions etc.
    - Accessible au programme appelant, **PUBLIC**
  - Corps ou **partie cachée ou implémentation ou BODY**
    - Corps des procédures ou des fonctions citées dans la partie spécification
    - Nouvelles procédures ou fonctions **privées** accessibles uniquement par des procédures ou fonctions du paquetage

Paquetage : *partie SPECIFICATION*

```
-- Partie spécification

CREATE [OR REPLACE] PACKAGE nom_package AS
Procedure Procedure1(liste des paramètres);
…
Function Fonction1(liste des paramètres)
                                  return TYPE;
…
Variable_globale1 type1;
…
CURSOR Curseur_global1 IS…
…
END nom_package;
/
```

Paquetage : *partie BODY*

```
-- Partie body

CREATE [OR REPLACE] PACKAGE BODY nom_package AS
PROCEDURE Procedure1(liste des paramètres)IS
…
BEGIN
…
END Procedure1;
FUNCTION Fonction1(liste des paramètres)
                                    RETURN type IS
…
BEGIN
…
RETURN(…);
END Fonction2;
END nom_package;
/
```

Paquetage : exemple

```
CREATE PACKAGE Produit AS

-- Procédure publique Insertion
PROCEDURE Inserer_Produit (vRefProd Produit.RefProd%TYPE,
vDesign Produit.Designation%TYPE, vCat Produit.Cat%TYPE,
vPrix Produit.Prix%TYPE, vStock Produit.Stock%TYPE,
vPrixAch Produit.PrixAch%TYPE);

-- Procédure publique Commande
PROCEDURE Commande(vRefComm Produit.RefComm%TYPE,
vRefProd Produit.RefProd%TYPE, vQte
LigneCommande.QtComm);

-- Fonction publique Prix

FUNCTION Prix_Moyen (vcat Produit.Cat%TYPE) RETURN NUMBER

END Produit;
/
```

Paquetage : exemple

```
CREATE PACKAGE BODY Employe AS
-------------
PROCEDURE Inserer_Produit (vRefProd Produit.RefProd%TYPE,
vDesign Produit.Designation%TYPE, vCat Produit.Cat%TYPE,
vPrix Produit.Prix%TYPE, vStock Produit.Stock%TYPE,
vPrixAch Produit.PrixAch%TYPE)IS
--déclarations
BEGIN
…..
END Insere_Produit;
-----------
PROCEDURE Commande(vRefComm Produit.RefComm%TYPE,
vRefProd Produit.RefProd%TYPE, vQte LigneCommande.QtComm)IS
--déclarations
BEGIN
…..
END Commande;
-----------
Prix_Moyen (vcat Produit.Cat%TYPE) RETURN NUMBER IS
--déclarations
BEGIN
…..
END Prix_Moyen;
```