Le langage PL/SQL

2ème Année Cycle d'ingénieur « *Informatique et Ingénierie des données* »

ENSA Khouribga

Pr. SOUSSI Nassima

Année Universitaire : 2021/2022

Plan du Cours

I. Introduction au langage PL/SQL

II. Eléments de programmation, variables et types en PL/SQL

III. Structures de contrôle

III. Interactions avec la BD

IV. Les curseurs

V. Les exceptions

VI. Les procédures & fonctions

Introduction au Langage PL/SQL

Pourquoi PL/SQL ?

–SQL est un langage **non procédural** ‰

– Les traitements complexes sont parfois difficiles à écrire si on ne peut utiliser des variables et des structures de programmation comme les boucles et les alternatives. ‰

=> On ressent vite le **besoin d'un langage procédural** pour lier plusieurs requêtes SQL avec des variables et dans les structures de programmation habituelles.

PL/SQL : Définition

– **PL/SQL (P**rocedural **L**anguage/**SQL)** : langage de programmation procédural et structuré pour développer des applications autour de bases de donnes relationnelles (SQL).

– **PL/SQL = Extension de SQL** : des requêtes SQL cohabitent avec les instructions procédurales (*boucles*, *conditions*, ...).

– Créer des traitements complexes destinés à être stockés sur le serveur de bases de données (objets serveur).

– PL/SQL est un langage propriétaire de Oracle.

PL/SQL : Caractéristiques

PL/SQL **combine** la puissance de manipulation de données d'SQL avec la puissance de traitement des langages procédurales:

- Vous pouvez contrôler le déroulement du programme avec des déclaration comme IF et LOOP.
- Vous pouvez déclarer des variables, définir des procédures et fonctions et faire la gestion des différentes exceptions,
- PL/SQL vous permet de décomposer des problèmes complexes en compréhensible code (ou procédures), et permet l'utilisation de ce code dans plusieurs applications.

PL/SQL : Utilisation

PL/SQL peut être utilisé sous **3 formes** :

- Un bloc de code exécuté comme une commande SQL, via un interpréteur standard (SQL*PLus)
- Un fichier de commande PL/SQL
- Un programme stocké (*procédure*, *fonction*, *package* ou *trigger*)

PL/SQL : Avantages

– Intégration complète du langage SQL

– Syntaxe très claire et un ensemble d'options qui assure une meilleure cohérence du code avec les données.

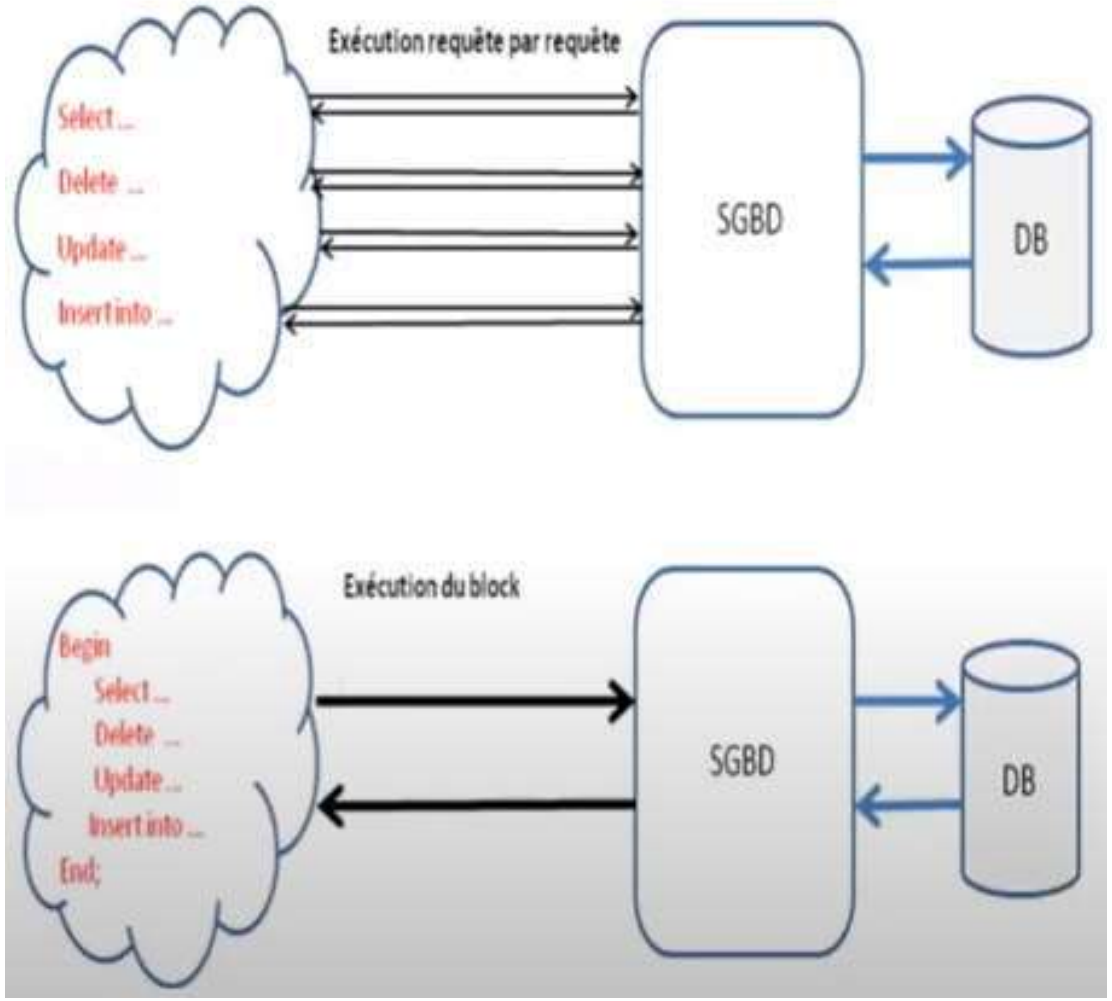
– Parfaite intégration avec Oracle et Java:

- On peut lancer des sous-programmes PL/SQL à partir d'un code Java.
- On peut appeler des procédures Java à partir d'un bloc PL/SQL.

PL/SQL vs. SQL

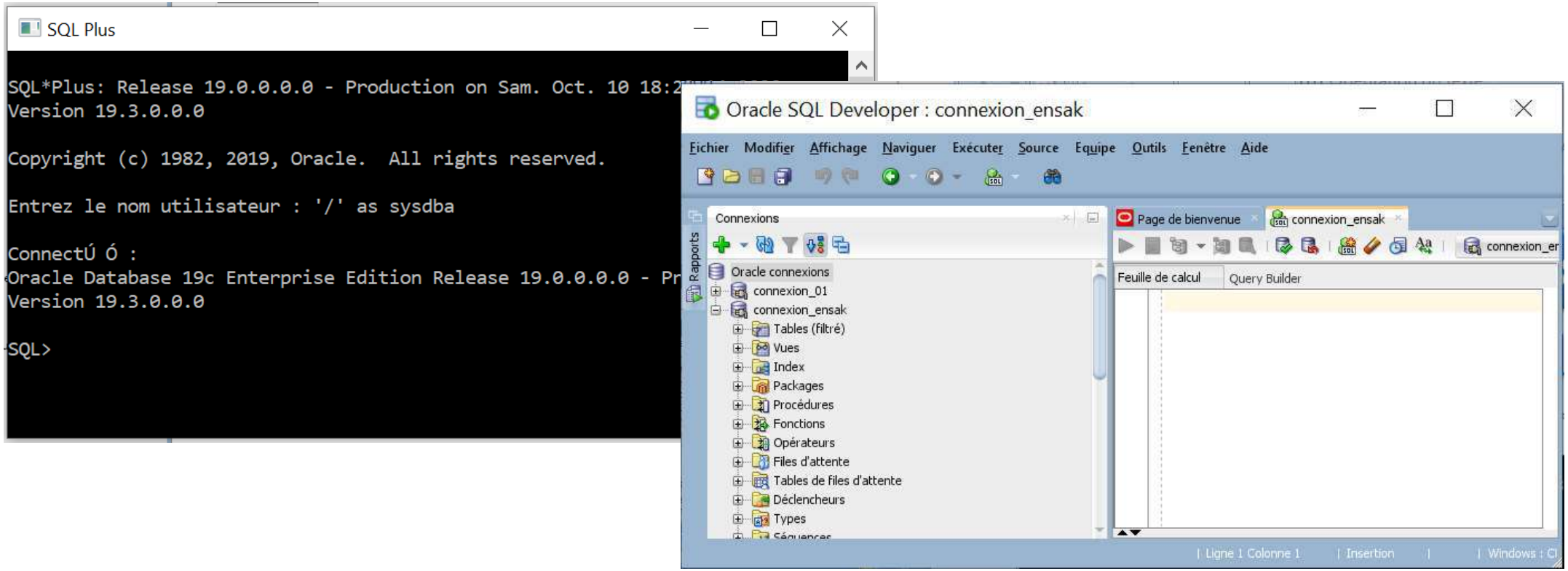
SQL	PL/SQL
C'est un Langage de requête structurée de base de données.	C'est un langage de programmation de base de données utilisant SQL.
Les variables de données ne sont pas autorisées	Les variables de données sont autorisées.
Aucune structure de contrôle prise en charge.	Les structures de contrôle sont prise en charges e.g la boucle for, while, etc.
La requête effectue une seule opération.	Le bloc PL/SQL exécute le groupe d'opérations en tant que bloc unique.
SQL est un langage déclaratif.	PL/SQL est un langage procédural.
Il est directement en interaction avec le serveur de base de données	N'interagit pas avec le serveur de base de données pour toutes les instructions.
C'est un langage orienté données.	C'est un langage orienté application.
Il est utilisé pour écrire des requêtes, des instructions DDL et DML.	Il s'agit de blocs de programme, de fonctions, de déclencheurs de procédures et de packages.

PL/SQL vs. SQL


Exécution requête par requête
Select ...
Delete ...
Update ...
Insert into ...
SGBD
DB
Exécution du block
Begin
Select ...
Delete ...
Update ...
Insert into ...
End;
SGBD
DB

Outils de développement

- Lien de téléchargement de la BD Oracle : cliquer ici
- Lien de téléchargement de SQL developper : cliquer ici



Eléments de programmation, variables et types en PL/SQL

Ordres SQL supportés dans PL/SQL

Ce sont les instructions du langage SQL :

Pour la manipulation de données:

- SELECT
- INSERT
- UPDATE
- DELETE

Et certaines instructions de gestion de transaction :

- COMMIT
- ROLLBACK
- SAVEPOINT
- LOCK TABLE
- SET TRANSACTION

Structure d'un programme PL/SQL

– PL/SQL est un langage structuré en **blocs**, constitués d'un ensemble **d'instructions**. On distingue 2 types de blocs :

- **Bloc anonyme** : bloc externe
- **Bloc nommé** : bloc stocké dans la base de données sous forme de *procédure*, *fonction* ou *trigger* (on lui attribue un nom)

– Un bloc anonyme contient trois parties :

1. une partie déclarative,
2. une partie exécutable,
3. une partie pour la gestion des exceptions.

Structure d'un bloc anonyme

DECLARE **(Facultatif)**

Déclarations des objets PL/SQL à utilisés dans ce bloc (variables, constantes, curseurs, ...)

BEGIN **(Obligatoire)**

Définir les instructions exécutables

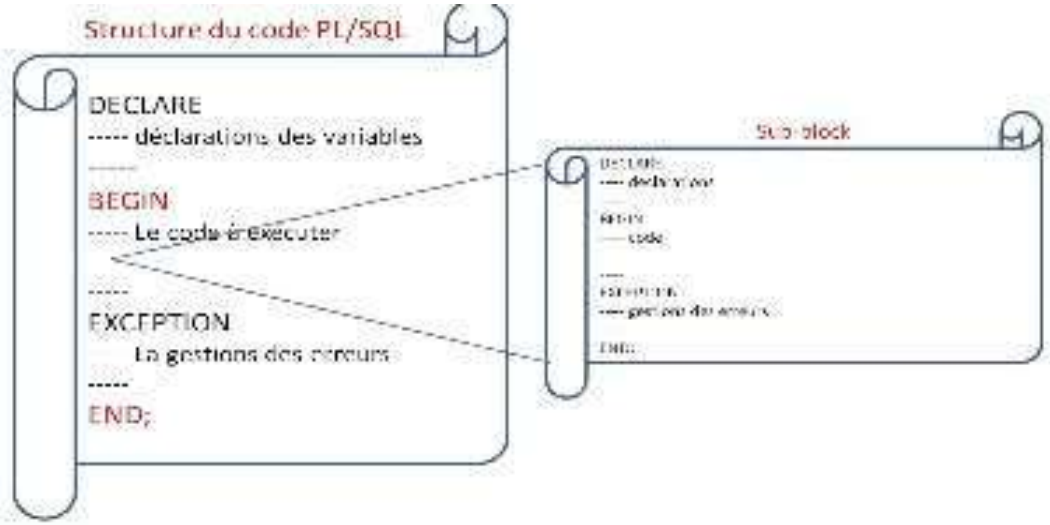
Exception **(Facultatif)**

Définir les actions à entreprendre en cas d'erreurs ou d'exceptions

END; **(Obligatoire)**

Imbrication d'un bloc anonyme

Un bloc peut être imbriqué dans le code d'un autre bloc (on parle de sous-bloc).



Les Variables en PL/SQL: Identificateur

Définition : les variables permettant l'échange d'information entre les requêtes SQL et le reste du programme.

- **Critère de définition d'un identificateur** sous Oracle:
  - 30 caractères au plus ,
  - commence par une lettre
  - peut contenir : lettres, chiffres, _, $ et #
  - insensible à la case
- Elles ont la portée habituelle des langages à blocs.
- Elles doivent être déclarées avant d'être utilisées.

Les Variables en PL/SQL : Conflit de noms

Si une variable porte le **même nom qu'une colonne** d'une table, c'est la colonne qui l'emporte :

```
DECLARE nom varchar(30) := 'Ahmed';
BEGIN
    DELETE FROM emp WHERE nom = nom;
END;
```

Solution : Pour éviter les conflits de nommage, préfixer les variables par **v_**

Les Variables en PL/SQL : Affectation

Plusieurs façons d'affecter une valeur à une variable :

1. Par « **:=** » v_name := 'Ahmed'
2. Par la directive **INTO** de la requête **SELECT**.

(Voir la section des « *Interactions avec la BD* »)

Les Variables en PL/SQL: Déclaration

⇒ **Syntaxe :**

***Nom_variable* [CONSTANT] *type_variable* [NOT NULL] [:= *valeur*];**

⇒ **Remarque :**

- Pour éviter les conflits de nommage, préfixer les variables par **v_**
- La déclaration multiple est **interdite : ~~(a, b integer;)~~**

⇒ **Exemples :**

```
v_DateNais      DATE;
v_NumDept       NUMBER(2) NOT NULL;
v_ville         VARCHAR(13) := 'Khouribga';
C_codeP         CONSTANT NUMBER := 25000;
```

Les Variables en PL/SQL: Types

– **Types habituels** : *integer, varchar, date, boolean, ...*

– **Types composites** : adaptés à la récupération des colonnes et des lignes des tables SQL :

- %TYPE
- %ROWTYPE

– **Types structurés** : en définissant un enregistrement ou un tableau PL/SQL.

Les Variables en PL/SQL: Types

- **Types composites : %TYPE**

Utiliser pour la déclaration d'une variable ayant le **même type** qu'une colonne d'une table (ou qu'une autre variable) :

Exemples :

```
v_nom        emp.nom%TYPE;
v_age        NUMBER(7);
v_min_age    v_age%TYPE := 10;
```

Les Variables en PL/SQL: Types

- **Types composites : %ROWTYPE**

Une variable peut **contenir** toutes les colonnes d'une ligne d'une table.

Exemple :

```
v_employe    emp%ROWTYPE;
```

=> On déclare que la variable ***v_employe*** contiendra une ligne de la table ***emp.***

Les Variables en PL/SQL: Types

- **Types structurés :: Enregistrement**

  – Il permet de déclarer un **ensemble de variables** à l'intérieur d'un enregistrement (Equivalent à ***STRUCT*** du langage C).

  Exemple :

```
TYPE record_etudiant IS RECORD
(  v_prenom          employees.prenom%TYPE ,
   v_niv_actuel      varchar(10) := 'IID2' ,
   v_date_inscrp     date := '10/09/2017'
);
etd_01 record_etudiant;
BEGIN
DBMS_OUTPUT.put_line('L'’étudiant ' || etd_01.v_prenom || ' a été inscrit en ' || etd_01.v_date_inscrp );
END;
```

Structures de Contrôles en PL/SQL

Structures Conditionnelles : IF

- **Objectif :** Test de condition simple
- **Syntaxe :** **IF** <condition> **THEN** <instruction(s)> **END IF;**
- **Exemples :**

```
DECLARE   dep      Number:=5;
          salaire  Number := 9000;
BEGIN
     IF dep = 3 THEN
        salaire:=salaire*1,15;
     END IF;
END;
```

```
DECLARE
dep     Number := 3;
salaire Number := 10000;
emp_nom VARCHAR(10):= 'Ahmed';
BEGIN
IF dep = 3 THEN
     IF emp_nom = 'Ahmed' THEN
          salaire:=salaire*1.15;
     END IF;
END;
```

Structures Conditionnelles : IF ELSE

– **Objectif :** Test de condition simple avec traitement de la condition opposée [*possibilité d'imbrication de plusieurs conditions*].

– **Syntaxe :**

IF <condition> **THEN**

<instruction(s)>

ELSE <instruction(s)>

END IF;

```
DECLARE
    dep     Number := 3;
    salaire Number := 10000;
BEGIN
    IF dep = 3 THEN
        salaire:=salaire*1,15;
    ELSE
        salaire:=salaire*1,05;
    END IF;
END;
```

Structures Conditionnelles : IF ELSIF

– **Objectif :** Test de plusieurs conditions

– **Syntaxe :**

```
IF <condition> THEN
      <instruction(s)>
ELSIF <condition>  THEN
      <instruction(s)>
ELSIF <condition>  THEN
       <instruction(s)>
…
ELSE …
END IF;
```

– **Exemple :**

```
…
IF dep = 3 AND emp_nom = 'Ahmed' THEN
     salaire:=salaire*1.15;
ELSIF dep = 4 THEN
     salaire:=salaire*1.18;
ELSE
   salaire:=salaire*1.05;
END IF;
…
```

Structures Conditionnelles : CASE

– **Objectif :** Test de plusieurs conditions

– **Syntaxe :**

```
CASE <variable>
WHEN <expression 1> THEN
        <valeur 1>
…
WHEN <expression n> THEN
        <valeur n>
ELSE …
END;
```

– **Exemple 1:**

```
DECLARE
resultat   VARCHAR(30);
Proprietaire VARCHAR(10);
BEGIN
resultat := CASE Proprietaire
        WHEN 'SYS' THEN ‘Le Propriétaire est SYS'
        WHEN 'NS' THEN ‘Le Propriétaire NS‘
        ELSE ‘Le Propriétaire est inconnu !!!'
            END;
    dbms_output.put_line(resultat);
END;
```

Structures Conditionnelles : CASE

– **Objectif :** Test de plusieurs conditions

– **Syntaxe :**

```
CASE <variable>
WHEN <expression 1> THEN
        <valeur 1>
…
WHEN <expression n> THEN
        <valeur n>
ELSE …
END;
```

– **Exemple 2 :**

```
DECLARE
resultat   VARCHAR(10);
Note       Number;
BEGIN
resultat := CASE
        WHEN  Note<8 THEN 'Ajourné'
        WHEN  Note<12 THEN 'Non Valide'
        ELSE 'Valider'
        END;
    dbms_output.put_line(resultat);
END;
```

Structures Itératives : LOOP

– **Objectif :** Exécution à plusieurs reprises d'un groupe d'instructions.

– **Syntaxe :**

```
LOOP
    <instruction(s)>;
    EXIT WHEN [condition] ;
    <instruction(s)>;
END LOOP;
```

⇒ La commande **EXIT** est obligatoire pour éviter une boucle infinie.

– **Exemple :**

```
DECLARE
cpt    Number := 0;
BEGIN
    LOOP
        cpt := cpt +1;
        dbms_output.put_line(cpt);
        EXIT WHEN cpt =10;
    END LOOP;
END;
```

Structures Itératives : WHILE

– **Objectif :** Exécution d'un groupe d'instructions jusqu'à vérification d'une condition.

– **Syntaxe :**

```
WHILE <condition> LOOP
    <instruction>;
    …
END LOOP;
```

– **Exemple :**

```
DECLARE
cpt    Number := 0;
BEGIN
    WHILE cpt <10 LOOP
        dbms_output.put_line(cpt);
        cpt := cpt +1;
    END LOOP;
END;
```

Structures Itératives : FOR

– **Objectif :** Itérations d'un groupe d'instructions un certain nombre de fois.

– **Syntaxe :**

FOR <variable d'itération> **IN** <borne inf>**..**<borne sup> **LOOP**
    <instruction(s)>;
END LOOP;

```
DECLARE
cpt    Number := 1;
BEGIN
    FOR cpt IN 1..10 LOOP
            dbms_output.put_line(cpt);
    END LOOP;
END;
```

Interactions avec la BD

Extraction et Affectation : SELECT INTO

Pour **extraire** les données dans un programme PL/SQL, on utilise l'instruction **SELECT** qui doit être utiliser **OBLIGATOIREMENT** avec l'instruction **INTO**.

=> La clause **INTO** permet de passer des valeurs d'une table dans des variables (*affectation*)

```
DECLARE
v_nom      Employees.nom%TYPE;
v_salaire  Employees.salaire%TYPE;
BEGIN
SELECT nom, salaire INTO v_nom, v_salaire
FROM  Employees
WHERE  matr = 01220;
…
END;
```

Extraction et Affectation : SELECT INTO

Sélectionner plusieurs enregistrements et les **affecter** à la variables *v_empRecord*:

```
DECLARE
    v_empRecord      Employees%RowType;
BEGIN
    SELECT * INTO v_empRecord
    FROM  Employees
    WHERE matr = 01220;
    dbms_output.put_line(v_empRecord.nom);
    END;
```

Extraction et Affectation : SELECT INTO

Remarques :

– Le **SELECT** ne doit renvoyer qu'une seule ligne.

⇒ Si le select renvoie plus d'une ligne, une exception (*TOO_MANY_ROWS*) est levée.

⇒ Si le select ne renvoie aucune ligne, une exception (*NO_DATA_FOUND*) est levée.

– Pour traiter des requêtes renvoyant plusieurs ligne, il faut utiliser les **Curseurs** (*Voir la section suivante*).

Mise à jour des Données : UPDATE

Pour mettre à jour le contenu d'une table, on utilise l'instruction **UPDATE**.

```
DECLARE
    v_aug_salaire employees.salaire%TYPE := 1500;
BEGIN
    UPDATE employees SET salaire = salaire + v_aug_salaire
    WHERE matr = 01220;
    COMMIT;
    …
END;
```

Mise à jour des Données : UPDATE

Remarques :

– Contrairement à l'instruction SELECT, si aucun enregistrement n'est modifié par l'instruction UPDATE, aucune erreur ne se produit et aucune exception n'est levée.

– l'instruction **COMMIT** valide toutes les modifications de la session en cours (*UPDATE, INSERT et DELETE*).

– Les affectations dans le code PL/SQL utilisent obligatoirement l'opérateur «:=», tandis que les comparaisons ou affectations SQL nécessite l'opérateur « = ».

Suppression de Données : DELETE

Pour supprimer les données d'une table dans la BD on utilise l'instruction **DELETE**.

```
DECLARE
    v_service employees.service%TYPE ;
BEGIN
    v_service := 'achat';
    DELETE FROM employees WHERE service = v_service ;
    COMMIT;
    …
END;
```

Les Curseurs

Définition d'un Curseur

– **Un curseur** est un espace mémoire qui contient le résultat d'une requête => toutes les requêtes SQL sont associées à un curseur.

– **Un curseur** est une variable permettant d'accéder à un résultat de requête SQL représentant une collection (*ensemble d'enregistrements*).

– On distingue deux types de curseur :

1. Implicite
2. Explicite

Types de Curseur : Implicite

– Les curseurs implicites ne sont pas déclarés par l'utilisateur.

– Ils sont déclarés et gérés automatiquement par le serveur Oracle lors de l'exécution d'une requête pour la tester et analyser.

– Ils sont associés aux ordres SELECT, INSERT, DELETE et UPDATE

⇒ Attention un seul enregistrement doit être résultat pour une requête SELECT

– Les curseurs implicites sont tous nommés **SQL**.

Types de Curseur : Implicite

=> Attributs de curseur implicite:

PL/SQL fournit des attributs permettant d'évaluer le résultat d'un curseur implicite :

Attribut	Description
SQL%ROWCOUNT	Entier : Nombre de lignes affectées par le dernier ordre SQL
SQL%FOUND	Booléen : TRUE si le dernier ordre SQL affecte au moins une ligne
SQL%NOTFOUND	Booléen : TRUE si le dernier ordre SQL n'affecte aucune ligne
SQL%ISOPEN	Toujours FALSE pour les curseurs implicites.

Types de Curseur : Implicite

⇒ **Exemple de curseur implicite :**

```
DECLARE
    Nb_lignes Integer;
BEGIN
        DELETE FROM employees WHERE dpt_num = 10 ;
        Nb_lignes := SQL%ROWCOUNT;
END;
```

Types de Curseur : Explicite

– Ils sont créés et gérés par l'utilisateur afin de pouvoir traiter un SELECT qui retourne **plusieurs lignes.**

– **Motivation :**

✓ Besoin de consulter des n-uplets issus d'une ou de plusieurs tables de la base de données.

✓ Effectuer des traitements en examinant chaque ligne individuellement.

Types de Curseur : Explicite

L'utilisation d'un curseur explicite nécessite 4 étapes :

1. **Déclaration du curseur**
2. **Ouverture du curseur**
3. **Traitements des lignes**
4. **Fermeture du curseur**

Curseur Explicite : Déclaration

– **Déclaration d'un curseur** ⇔ Association d'un nom de curseur à une requête SELECT

– Elle se fait dans la section ***DECLARE*** d'un bloc PL/SQL :

CURSOR nom-curseur **IS** requête_sql ;

Exemple :

```
DECLARE
    CURSOR C1 IS SELECT employe_id, nom FROM employees;
    CURSOR C2 IS SELECT * FROM employees
                 WHERE departement_id = 29;
BEGIN
    …
END;
```

Curseur Explicite : Ouverture

– Après avoir déclaré le curseur, il faut l'ouvrir dans la section exécutable.

OPEN nom_curseur **;**

⇒ **Conséquences :**

- allocation mémoire du curseur
- analyse et exécution de l'instruction SELECT

Curseur Explicite : Fermeture

– La fermeture d'un curseur consiste à la libération de la zone qui devient inaccessible.

CLOSE nom_curseur ;

– Fermer le curseur après avoir terminé le traitement de lignes.

– Ne pas essayer d'extraire les données d'un curseur s'il a été fermé.

Curseur Explicite : Traitement des lignes

– Après l'exécution du SELECT, les lignes récupérées sont traitées d'une manière séquentiel.

– La valeur de chaque ligne du SELECT doit être stockée dans une variable réceptrice.

FETCH nom_curseur **INTO** liste_variables;

– FETCH ramène une seule ligne

– Pour traiter n lignes, prévoir une boucle.

Curseur Explicite : Traitement des lignes

```
DECLARE
    v_empId   employee.employee_id%TYPE ;
    v_nom  employee.nom%TYPE ;
    CURSOR C1 IS SELECT employee_id, nom FROM employee;
BEGIN
    OPEN C1;
    FOR i IN 1..10 LOOP
        FETCH C1 INTO v_empId, v_nom;
        …
    END LOOP;
    CLOSE C1;
END;
```

Attributs sur les curseurs

- **Indicateurs sur l'état d'un curseur :** *nom_curseur%attribut*
  - **%FOUND** : booléen VRAI si un n-uplet est trouvé
  - **%NOTFOUND** : booléen VRAI après la lecture du dernier n-uplet
  - **%ISOPEN** : booléen VRAI si le curseur est actuellement actif
  - **%ROWCOUNT** : numérique, retourne le nombre total de n-uplets renvoyées jusqu'à présent.
- **Remarque :**
  - Curseur ***implicite*** : **SQL%FOUND**, ….
  - Curseur ***explicite*** : **nom_curseur%FOUND**, …

Curseur Explicite : Exemple 1

```
DECLARE
v_nom     employees.nom%TYPE ;
v_salaire  employees.salaire%TYPE ;
CURSOR C_dpt_10 IS SELECT nom, salaire FROM employees WHERE dpt_num=10 ;
BEGIN
OPEN C_dpt_10 ;
LOOP
    FETCH C_dpt_10 INTO v_nom, v_salaire ;
    exit when C_dpt_10 %NOTFOUND ;
    IF v_salaire > 8000 THEN
       INSERT INTO resultat VALUES (v_nom, v_salaire) ;
    END IF;
     DBMS_OUTPUT.put_line('nom = ' || v_nom || ' salaire = ' || v_salaire);
END loop ;
CLOSE C_dpt_10 ;
END;
```

Curseur Explicite : Exemple 2

```
…
BEGIN
OPEN C_dpt_10 ;
LOOP
    FETCH C_dpt_10 INTO v_nom, v_salaire ;
    IF C_dpt_10%FOUND THEN
            DBMS_OUTPUT.PUT_LINE(dpt_10%ROWCOUNT);
            DBMS_OUTPUT.put_line('nom = ' || v_nom || ' salaire = ' || v_salaire);
    ELSE EXIT;
    END IF;
END loop ;
CLOSE C_dpt_10 ;
END;
```

Les Curseurs Paramétrés

```
DECLARE
CURSOR nom_curseur (p1 type_p1, p2 type_p2,...) IS select_query ;
BEGIN
OPEN nom_curseur(val1, val2,...) ;
...
CLOSE nom_curseur;
```

Remarques :

- Type : char, number, date, boolean SANS spécifier la longueur
- Passage des valeurs des paramètres **juste** à l'ouverture du curseur

Les Curseurs Paramétrés : Exemple

```
DECLARE
    CURSOR curs(p_dep  employees.dpt_num%TYPE) IS SELECT nom,salaire
    FROM employees WHERE dpt_num = p_dep ;
BEGIN
    OPEN curs(10);
    LOOP
        FETCH curs INTO v_nom, v_salaire ;
        EXIT WHEN curs%NOTFOUND ;
        DBMS_OUTPUT.put_line('nom = ' || v_nom || ' salaire = ' || v_salaire);
    END LOOP;
    CLOSE curs;
END;
```

Les Exceptions

Les Exceptions : Définition

– Une exception est un avertissement ou une erreur rencontré(e) lors de l'exécution.

– Le language PL/SQL offre aux développeurs un mécanisme de gestion des exceptions.

⇒ Il permet de préciser la **logique du traitement des erreurs** survenues dans un bloc PL/SQL.

⇒ Il permet aussi de **protéger l'intégrité du système**.

Les Exceptions : Types

– Il existe deux types d'exceptions :

- **Exception externe (*utilisateur*) :** causée par le programme utilisateur (déclarés par l'utilisateur)
- **Exception interne** au SGBD (erreur système)

Ex : espace mémoire insuffisant, mémoire saturée, table inexistante, connexion non établie, division par zéro, …

- **Exceptions anonymes**

Exception utilisateur : définition & gestion

– Traitement d'une erreur utilisateur survenue dans un bloc PL/SQL :
  - Définir et donner un nom à chaque erreur.
  - Déclarer le nom de l'erreur dans la partie DECLARE.
  - Associer et définir le traitement spécifique à effectuer pour l'erreur à la section EXCEPTION.

– **Exemples de traitements :**
  Notification à l'utilisateur, Annulation de l'opération, …

Exception utilisateur : Déclaration

```
DECLARE
    ...
    nom_exception EXCEPTION ;
    ...
BEGIN
    …
    IF (anomalie) THEN RAISE nom_exception;
    ...
    EXCEPTION
    WHEN nom_erreur1 THEN (traitement_1)
    WHEN nom_erreur2 THEN (traitement_2)
    …
    [WHEN OTHERS THEN traitement_N]
END;
```

Exception utilisateur : Exemple

Afficher les informations d'un employée sélectionné par ID saisi par l'utilisateur.

```
DECLARE
      emp_id   employees.matr%type := &id_saisi;
      Id_invalid EXCEPTION;
BEGIN
      IF emp_id <=0 then RAISE Id_invalid ;
      ELSE
        dbms_output.put_line('on selectionne l employée demandé');
      END IF;
      EXCEPTION
      WHEN Id_invalid THEN
         dbms_output.put_line('L id doit etre supérieur strictement à 0');
END;
```

Exception interne (système)

– **CURSOR_ALREADY_OPEN** : tentative d'ouverture d'un curseur déjà ouvert.

– **ZERO_DIVIDE** : Division par 0

– **INVALID_CURSOR** : Tentative d'accès à un curseur non ouvert.

– **INVALID_NUMBER** : Utilisation d'un type non numérique dans un contexte où un nombre est requis.

– **NO_DATA_FOUND** : SELECT... INTO ne retourne aucun résultat

– **NOT_LOGGED_ON** : Tentative d'exécution d'opération SQL sans être connecté à Oracle.

– **STORAGE_ERROR** : Erreur de stockage

Exception interne (système)

– **VALUE_ERROR** : Erreur de conversion arithmétique, contrainte de taille, etc.

– **INVALID_CURSOR** : opération incorrecte sur un curseur (Ex: fermeture d'un curseur qui n'est pas été ouvert).

– **LOGON_DENIED** : mauvais login/password lors de la connexion

– **ROWTYPE_MISMATCH** : types de paramètre incompatibles

– **TOO_MANY_ROWS** : trop de lignes renvoyées par un SELECT... INTO

– …

Exception interne (système)

EXCEPTION PRÉDÉFINIE	ERREUR ORACLE	SQLCODE
ACCESS_INTO_NULL	ORA-06530	-6530
CASE_NOT_FOUND	ORA-06592	-6592
CURSOR_ALREADY_OPEN	ORA-06511	-6511
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NO_DATA_NEEDED	ORA-06548	-6548
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
STORAGE_ERROR	ORA-06500	-6500
SYS_INVALID_ROWID	ORA-01410	-1410
TOO_MANY_ROWS	ORA-01422	-1422
ZERO_DIVIDE	ORA-01476	-1476

- Les erreurs ORACLE générées par le noyau sont numérotées (**ORA-**xxxxx).
- Vous pouvez retrouver l'ensemble des erreurs internes d'Oracle dans la documentation.

Exception interne (système) : Exemple

```
…
BEGIN
IF emp_id <=0 THEN RAISE Id_invalid ;
ELSE
    SELECT nom,dpt_num INTO v_nom, v_dpt FROM employees WHERE matr = emp_id;
  dbms_output.put_line('nom = ' || v_nom || '--- departement : ' || v_dpt);
END IF;
EXCEPTION
WHEN Id_invalid THEN dbms_output.put_line('L id doit etre supérieur strictement à 0');
WHEN NO_DATA_FOUND THEN dbms_output.put_line('l employé demandé n existe pas');
END;
```

Exceptions Anonymes : Définition

– Pour les codes d'erreur n'ayant pas de nom associé, il est **possible de définir un nom d'erreur**.

– Ce type d'exceptions crée une **correspondance** entre le **code erreur ORACLE** et le **nom de l'exception** (ce nom est choisi librement par le programmeur).

– Pour affecter un nom à un code d'erreur, on doit utiliser une directive compilée (PRAGMA) nommée EXCEPTION_INIT.

Exceptions Anonymes : Syntaxe

```
DECLARE
nom_erreur     EXCEPTION ;
PRAGMA     EXCEPTION_INIT(nom_erreur, code_erreur);
BEGIN
    ...
    dès que l'erreur Oracle est rencontrée, passage automatique à
    la section EXCEPTION pour réaliser le traitement approprié ...
    ...
    EXCEPTION
    WHEN nom_erreur THEN (traitement) ;
    [WHEN OTHERS THEN (traitement) ;] ;
END;
```

Exceptions Anonymes : Exemple

```
DECLARE
    aucune_donnees EXCEPTION;
    PRAGMA EXCEPTION_INIT (aucune_donnees, 100);
    BEGIN
     SELECT nom,dpt_num INTO v_nom, v_dpt FROM employees WHERE matr = emp_id;
    dbms_output.put_line('nom = ' || v_nom || '--- departement : ' || v_dpt);
    EXCEPTION
    WHEN aucune_donnees THEN
            dbms_output.put_line(‘l employé n existe pas');
END;
```

Les Procédures

Bloc anonyme ou nommé

– Un bloc anonyme PL/SQL est un bloc :

DECLARE ...BEGIN ...END

comme dans les exemples précédents.

– On peut exécuter directement un bloc PL/SQL anonyme.

– Un bloc nommé est définie par une procédure ou une fonction pour réutiliser le code.

Procédure : Définition

– Une procédure est un type de sous-programme qui exécute une action.

– Une procédure peut être **stockée** en tant qu'objet dans la base de données en vue d'exécutions répétées (*en l'appelant par son nom*).

– Une procédure est compilée avant l'exécution du programme.

– Les procédures peuvent être utilisées dans d'autres procédures ou fonctions ou dans des blocs PL/SQL anonymes.

Procédure : Utilité

– **Masquer la complexité du code PL/SQL** : simple appel de procédure avec passage d'arguments.

– **Garantir l'intégrité des données** : encapsulation des données par les procédures.

– **Sécuriser l'accès aux données** : accès à certaines tables seulement à travers les procédures.

– **Optimiser le code** : une procédure peut être exécutée par plusieurs utilisateurs.

Procédure : Syntaxe générale

```
CREATE [OR REPLACE]  PROCEDURE  nom_procédure
[(<liste de paramètres>)] IS
[ -- Déclarations des variables]
BEGIN
      -- Corps de la procédure
END [nom_procédure];
```

- L'option **REPLACE** indique que, si la procédure existe, elle sera supprimée et remplacée par la nouvelle version créée avec l'instruction.
- **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 ».

Procédure : Syntaxe générale

Déclaration des paramètres :

Nom_paramètre **[IN | OUT | IN OUT]** type_paramètre
[:= | DEFAULT expression]

- **Type_paramètre** : un type PL/SQL.
- **IN** : paramètre en entrée, non modifié par la procédure (par défaut : IN)
- **OUT** : paramètre en sortie, peut être modifié par la procédure, transmis au programme appelant.
- **IN OUT** : à la fois en entrée et en sortie.

Procédure : Exemple 1

Exemple 1 : Paramètre IN

```
CREATE PROCEDURE  ajouter_emp( p_nom IN VARCHAR, p_salaire IN NUMBER) IS
BEGIN
   INSERT  INTO  resultat(reslt_nom, reslt_salaire) VALUES(p_nom, p_salaire);
END ;
```

Procédure : Exemple 2

Exemple 2 : Paramètre OUT

```
CREATE  PROCEDURE  select_emp(p_id  IN employees.matr%TYPE,
p_nom      OUT employees.nom%TYPE,
p_salaire  OUT employees.salaire%TYPE,
p_dep      OUT employees.dpt_num%TYPE)
IS
BEGIN
    SELECT nom, salaire, dpt_num INTO p_nom, p_salaire, p_dep
    FROM employees WHERE matr = p_id;
END ;
```

Procédure : Exemple 2

- **Appel via un Bloc PL/SQL :**

```
DECLARE
    v_nom       employees.nom%TYPE;
    v_salaire   employees.salaire%TYPE;;
    v_dep       employees.dpt_num%TYPE;
BEGIN
    select_emp(1220, :v_nom, :v_salaire, :v_dep);
    dbms_output.put_line('l employé sélectionner est nommé :' || v_nom );
END;
```

Remarque : Par défaut, le paramètre IN est transmis en valeur, tandis que le paramètre OUT et IN OUT est transmis par référence.

Procédure : Exemple 2

- **Appel via SQL*Plus**:

```
VARIABLE   v_nom      VARCHAR2(20);
VARIABLE   v_salaire   NUMBER;
VARIABLE   v_dep       NUMBER ;
EXECUTE   select_emp(1220, :v_nom, :v_salaire, :v_dep);
PRINT v_nom;
```

Remarque : Sous SQL*PLUS, il faut taper une dernière ligne contenant « / » pour compiler une procédure ou une fonction (*si vous la déclarez en SQL*PLUS*).

Procédure : Exemple 3

Exemple 3 : Paramètre IN OUT

```
CREATE   OR REPLACE PROCEDURE  new_format_phone
(p_phone_num  IN OUT VARCHAR)
IS
BEGIN
      p_phone_num := '(' || SUBSTR(p_phone_num,1,3) ||
                     ')' || SUBSTR(p_phone_num,4,3) ||
                     '-' || SUBSTR(p_phone_num,7,6) ;
END new_format_phone;
```

Procédure : Exemple 3

- **Appel via un BlocPL/SQL :**

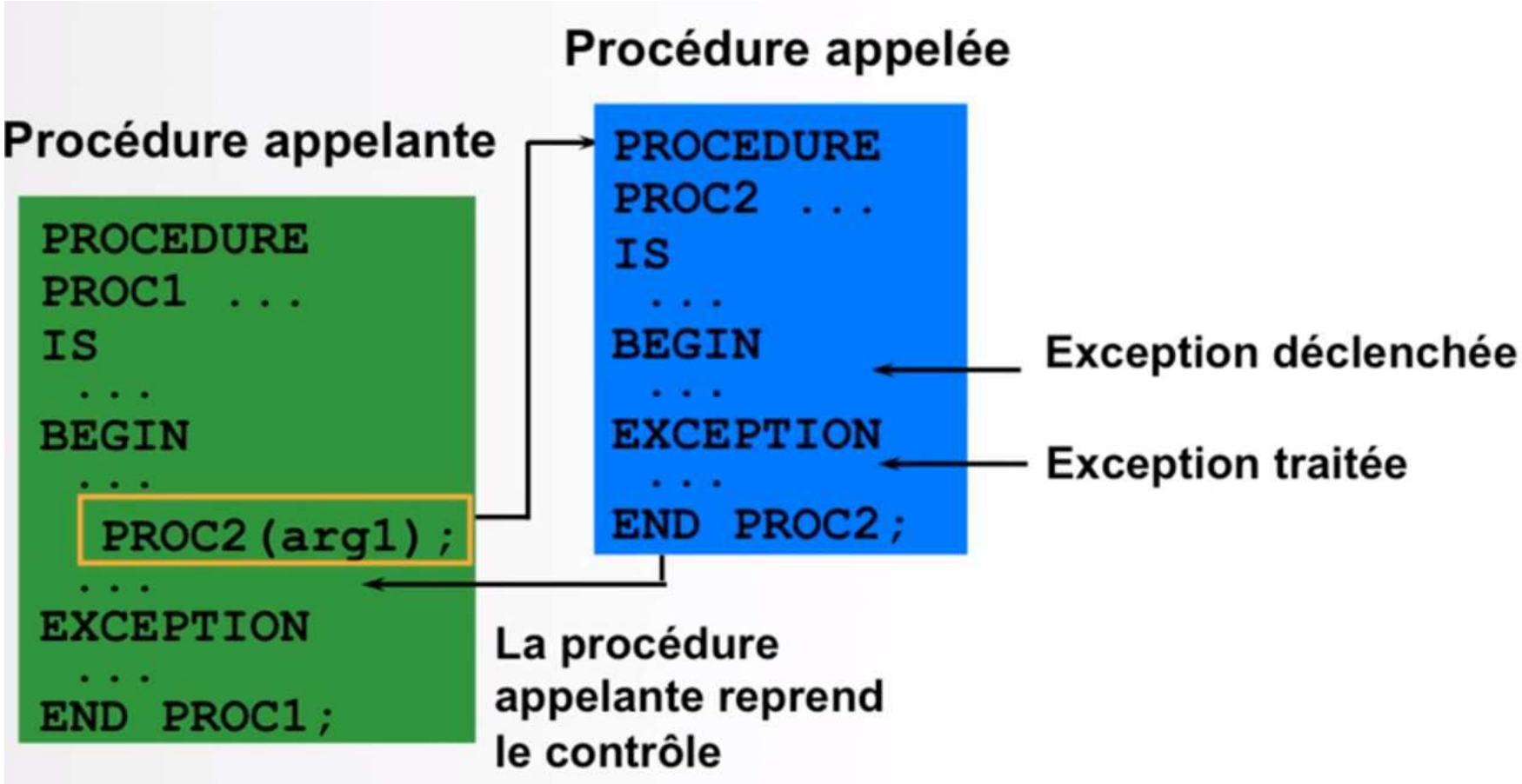
```
DECLARE
    tel  VARCHAR(15) := '212601020304' ;
BEGIN
    new_format_phone(tel);
    dbms_output.put_line('le nouveau format de num est : ' || tel);
END;
```

Procédure : Exemple 3

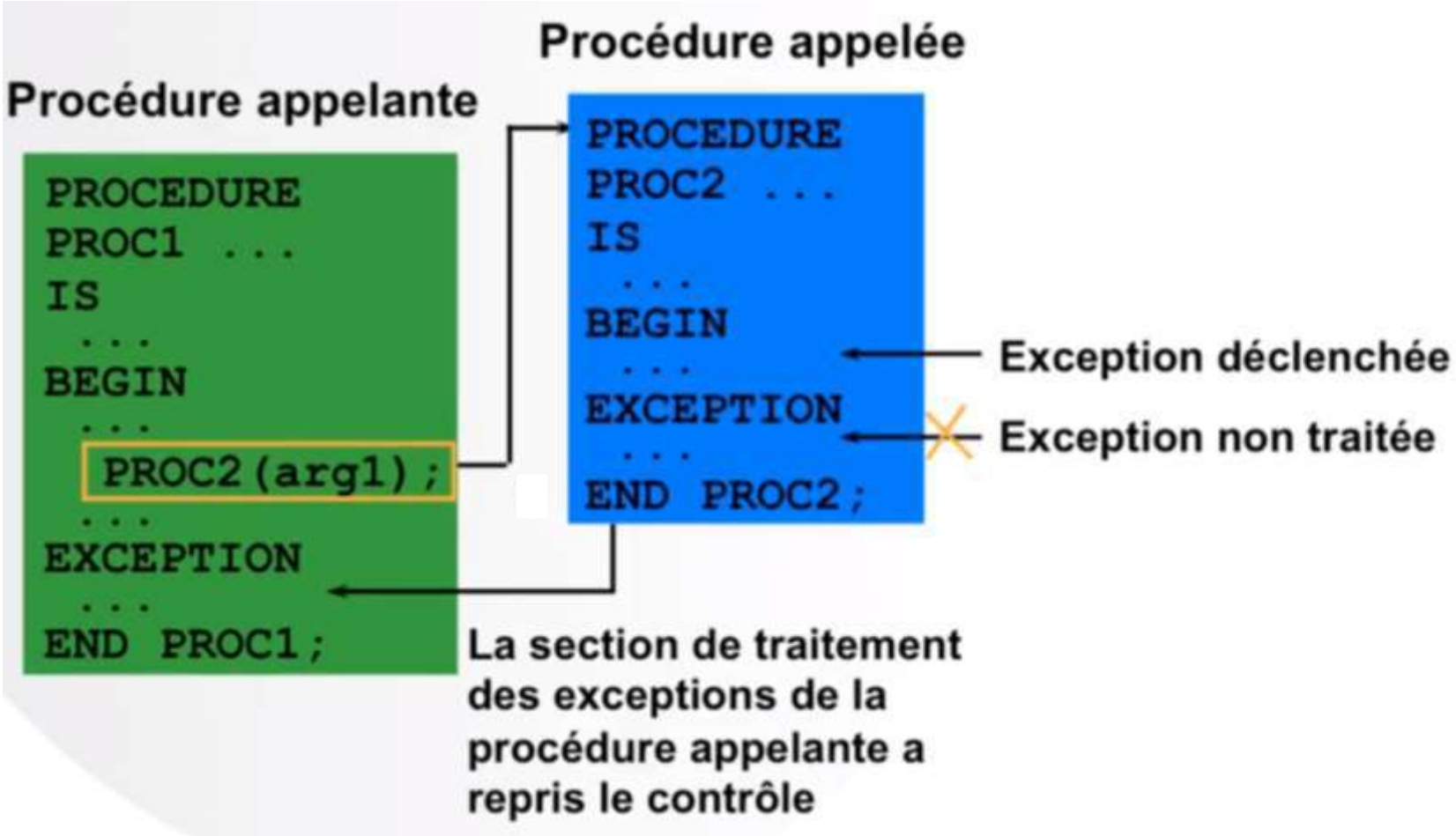
- **Appel via SQL*Plus**

```
VARIABLE tel VARCHAR(15)
EXECUTE :tel := '212601020304' ;
EXECUTE new_format_phone(:tel);
PRINT tel;
```

Procédure : Gestion des exceptions



Procédure : Gestion des exceptions



Procédure : Suppression

- **Syntaxe de suppression:**

```
DROP   PROCEDURE    nom_procédure ;
```

- **Exemple :**

```
DROP   PROCEDURE    new_format_phone ;
```

Les Fonctions

Fonction : Définition

– Une fonction est une **procédure** retournant une valeur.

– Une fonction peut être **stockée** en tant qu'objet de schéma dans la base de données en vue d'exécutions répétées.

– Une fonction peut aussi être utilisée dans les requêtes SQL, dans d'autres procédures ou fonctions ou dans des blocs PL/SQL anonymes.

Fonction : Syntaxe de déclaration

```
CREATE  [OR REPLACE]  FUNCTION nom_fonction
[(<liste de paramètres>)] RETURN  <Type_de_retour> IS
[ ---Déclarations des variables]
BEGIN
    -- Corps de la fonction
    RETURN valeurRetour;
END [nom_fonction];
```

Remarque :

- La fonction doit contenir au moins une instruction RETURN.
- Le type de retour ne doit pas inclure la spécification de taille.

Fonction : Exemple

```
CREATE FUNCTION Mad_to_Pound (somme IN NUMBER)
RETURN NUMBER IS
taux CONSTANT NUMBER := 0.08419;
BEGIN
    RETURN somme*taux;
END;
```

Remarque : Pour les fonctions, seul le passage par valeur (**IN**) est autorisé.

Fonction : Etapes d'exécution

1. Appeler une fonction dans une expression PL/SQL (*requête SQL*, *procédures*, *fonctions*, *bloc PL/SQL*)
2. Créer une variable destinée à recevoir la valeur renvoyée.
3. Exécuter la fonction.
4. La valeur renvoyée par l'instruction **RETURN** sera placée dans la variable.

Fonction : Appel & exécution

- **Exemple :** via une requête SQL

```
SELECT emp_id, nom, salaire AS salaireMAD,
Mad_to_Pound(salaire) AS salairePd
FROM employees WHERE dept_id = 29;
```

⇒ **Remarque :**

Vous pouvez appeler les fonctions dans les emplacements SQL suivants : *SELECT, WHERE, HAVING, ORDER BY, GROUP BY, VALUES de la commande INSERT et SET de la commande UPDATE*

Fonction : Suppression

- **Syntaxe de suppression:**

```
DROP   FUNCTION    nom_fonction ;
```

- **Exemple :**

```
DROP    FUNCTION   Mad_to_Pound ;
```

Procédure vs. Fonction

Procédure	Fonction
Exécuter une action	Calculer une valeur
S'exécute en tant qu'instruction PL/SQL	Sont appelés dans une expression
Peut transférer zéro, une ou plusieurs valeurs	Doit renvoyer une seule valeur
Type de passage de paramètres : IN, OUT et IN OUT	Seul le passage par valeur (IN) est autorisé
Peut être utilisées dans : procédure, fonction ou un bloc PL/SQL anonymes	Peut aussi être utilisées dans les requêtes SQL