UNIVERSITÉ ASSANE SECK DE ZIGUINCHOR
UFR DES SCIENCES ET TECHNOLOGIES
DÉPARTEMENT D'INFORMATIQUE

CHAPITRE II
PL-SQL SOUS MYSQL

LICENCE 2 INGÉNIERIE INFORMATIQUE

ANNÉE ACADÉMIQUE 2021 – 2022

SEMESTRE 4

DR SERIGNE DIAGNE

PLAN DU COURS

Introduction

I. La structure d'un programme
   1. Configuration du serveur de données
   2. L'implémentation de bases de données

II. Les bases du PLSQL
   1. Affectation
   2. Opérateurs et commentaires
   3. Instructions conditionnelles
   4. Instructions itératives

III. Les triggers

IV. Les routines
   1. Procédure stockée
   2. Fonction

V. Les curseurs

INTRODUCTION

- ✓ Le langage SQL n’est pas un langage procédural. Il ne permet, donc pas :
  - de déclarer des variables ;
  - de créer des sous-programmes ;
  - de programmer l'exécution d'une action suivant un événement ;
  - de répéter l'exécution d'une ou plusieurs instructions.
- ✓ Un langage procédural, repose sur des procédures, des fonctions, des sous-routines, etc. ;
- ✓ Il spécifie toutes les étapes que l'ordinateur doit suivre pour atteindre l'état ou la sortie souhaité ;
- ✓ Il sépare un programme au sein de variables, fonctions, instructions et opérateurs conditionnels.

INTRODUCTION

- ✓ Ainsi, le langage PL/SQL (Procedural Language/SQL) est créé pour permettre de faire de la programmation procédurale sous MySQL et Oracle ;
- ✓ Il est basé sur le langage SQL donc il emprunte les types, les opérateurs, etc.

I. STRUCTURE D'UN PROGRAMME PLSQL

Un programme PL/SQL constitué de 3 composants dans deux parties :

- ✓ Les composants :
  - L'entête ;
  - Les déclarations ;
  - Les instructions.
- ✓ Les parties :
  - L'entête ;
  - corps du programme.

I. STRUCTURE D'UN PROGRAMME PLSQL

I. 1. L'ENTÊTE

- ✓ L'entête d'un module PL/SQL :
  - contient le type de programme ;
  - son nom ;
  - ses arguments éventuels (fonction et procédure) ;
  - le domaine de la valeur de retour (fonction) ;
  - le moment et l'évènement ainsi que la table sur laquelle elle porte (trigger) ;
  - etc.

I. STRUCTURE D'UN PROGRAMME PLSQL

I. 2. LES DÉCLARATIONS

- ✓ Les déclarations sont faites dans le corps du programme entre Begin et End ;
- ✓ C'est dans cette partie que toutes les déclarations sont faites : variable, curseur, gestionnaire, etc.
- ✓ Chaque ligne de déclaration est ouverte par le mot clé **Declare**.

Exemple :

Declare id_variable Domaine ;

I. STRUCTURE D'UN PROGRAMME PLSQL

I. 3. LES INSTRUCTIONS

- ✓ Les instructions sont écrites dans le corps du programme PL/SQL qui est ouvert par le mot-clé **Begin** et fermé par le mot-clé **End**.
- ✓ Toutes les instructions à exécuter dans le programme doivent être écrites entre ces deux mots-clés.

Exemple :

```
Begin
        Instruction_1 ;
        Instruction_2 ;
End ;
```

II. LES BASES DU PLSQL

II.1. L'Affectation

Il existe trois types d'affectation sous PL-SQL :

- ✓ **Initialisation pendant la déclaration :**

Declare id_variable type **Default** Valeur ;

- ✓ **Affectation d'une valeur scalaire dans le programme :**

Set id_variable = valeur ;

- ✓ **Affectation du résultat d'une requete**

Select Attribut **Into** id_variable **From** Table **Where** condition(s) ;

II. LES BASES DU PLSQL

II.1. L’Affectation

Exemple :

- ✓ **Declare** a integer Default 25 ;
- ✓ **Set** a = 15 ;
- ✓ **Select** Age **Into** a **From** Etudiant **Where** Matricule = 'MA00254' ;

II. LES BASES DU PLSQL

II.2. Les opérateurs

- ✓ Opérateurs ensemblistes : Not ; Is Null ; Like ; Between ; In ; Any
- ✓ Opérateurs booléens : And ; Or
- ✓ Opérateurs athmetiques : + ; - ; * ; / ; @ ; = ; <> ; <= ; >=

II. LES BASES DU PLSQL

II.3. Les commentaires

- ✓ Commentaires sur une ligne : -- Sur une seule ligne
- ✓ Commentaires sur plusieurs lignes : /* Sur plusieurs ligne */

II. LES BASES DU PLSQL

II.4. Les instructions conditionnelles

a. L'instruction IF

```
If  condition  Then
        Instruction (s) ;
ElseIf  condition Then
        Instruction(s) ;
Else
        Instruction(s) ;
End If ;
```

II. LES BASES DU PLSQL

II.4. Les instructions conditionnelles

Exemple :

```
Declare a integer ;
Declare b varchar(50) ;
Begin
        Select Count(*) Into a From Etudiant Where Age = 22 ;
        If a = 5 then
        Set b = 'Il y a cinq étudiants ayant 22 ans' ;
        ElseIf a = 10 Then
        Set b = 'Il y a dix étudiants ayant 22 ans' ;
        End If ;
        Select b ;
End ;
```

II. LES BASES DU PLSQL

II.4. Les instructions conditionnelles

b. L'instruction Case

```
Case  id_variable
        When Valeur_1 Then Instruction(s) ;
        When Valeur_2 Then  Instruction(s) ;
        -  -  -  -  -  -  -
        When Valeur_n Then Instruction(s) ;
        Else     Instruction(s) ;
End Case ;
```

II. LES BASES DU PLSQL

II.4. Les instructions conditionnelles

b. L'instruction Case

```
Case
        When Condition_1 Then Instruction(s) ;
        When Condition_2 Then Instruction(s) ;
          - - - - - - - - - - -
        When Condition_n Then Instruction(s) ;
        Else   Instruction(s) ;
End Case ;
```

II. LES BASES DU PLSQL

II.5. Les instructions itératives

While Condition **Do**

Instruction(s) ;

End While ;

Label : **Loop**

Instruction(s) ;

Leave Label ;

End Loop Label ;

Repeat

Instruction (s) ;

Until Condition ;

End Repeat ;

II. LES BASES DU PLSQL

II.5. Les instructions itératives


```
Declare a integer ;
Declare b boolean Default False ;
Begin
        Select Count(*) Into a From Etudiant Where VilleNaissance = 'Bignona' ;
        While a < 5 Do
                Set a = a + 1 ;
                Set b = True ;
        End While ;
        Select b ;
End ;
```

II. LES BASES DU PLSQL

II.5. Les instructions itératives

```
Declare a, c integer Default 0 ;
Begin
        Select Count(*) Into a From Etudiant Where VilleNaissance = 'Bignona' ;
        Iterloop : Loop
                If a >= 35 Then
                        Leave iterloop ;
                End If ;
                Set a = a + 5 ;
                Set c = c + 1 ;
        End Loop interloop ;
        Select c ;
End ;
```

II. LES BASES DU PLSQL

II.5. Les instructions itératives

Remarques :

- ✓ La commande **Leave** permet d’arrêter l'exécution de la boucle dans laquelle elle est placée ;
- ✓ La commande **Iterate** permet d'arrêter l'exécution en cours et de passer à une nouvelle.

II. LES BASES DU PLSQL

II.6. Variable globale

- ✓ Une variable globale est une variable dont l’identificateur est précédé du caractère '@'.
- ✓ Elle est visible dans le trigger ou la routine dans lequel elle est déclarée et dans les autres qui seront créées par la suite.
- ✓ Une variable qui n'est pas précédée de ce signe est donc une variable locale.

III. LES TRIGGERS

III. 1. Qu’est-ce qu’un trigger ?

- ✓ Un trigger est un objet de base de données associé à une table ;
- ✓ Il est appelé lorsqu'un événement particulier survient ;
- ✓ Son exécution est déclenchée par la survenance d’une action bien déterminée (Evénement) à un moment bien déterminé (Moment) ;

III. LES TRIGGERS

III. 1. Qu’est-ce qu’un trigger ?

Les triggers doivent respecter les conditions suivantes :

- ✓Un trigger ne peut être placé sur une vue ou une table temporaire ;
- ✓Deux triggers de même moment et même événement ne peuvent pas porter sur la même table ;
- ✓Il est nécessaire d’être super-utilisateur pour créer, modifier ou supprimer un trigger ;
- ✓Si le trigger échoue et que son instant d’exécution est BEFORE, l’action qui suit ne sera pas exécutée ;
- ✓La seule manière d’interrompre un trigger est de provoquer une erreur en son sein.

III. LES TRIGGERS

III. 2. Création d’un trigger

La syntaxe de la création d'un trigger est :

Create TRIGGER Nom_Trigger Moment_Trigger Evenement_Trigger
On Nom_Tabble **For Each Row**

Declare

Begin

Instruction (s) ;

End ;

III. LES TRIGGERS

III. 2. Création d’un trigger

Remarques :

- ✓ **Temps_Trigger** est soit **Before** soit **After**.
- ✓ **Evenement_Trigger** est soit **Insert** soit **Update** soit **Delete**
- ✓ Dans le corps d'un trigger, on peut faire référence aux colonnes dans la table associée en utilisant les mots **OLD** et **NEW**.
  - **OLD.*col_name*** fait référence à une colonne d'une ligne existante avant sa modification ou sa suppression ;
  - **NEW.*col_name*** fait référence à une colonne d'une ligne après insertion ou modification.

III. LES TRIGGERS

III. 2. Création d’un trigger

Remarque :

- ✓ L'utilisation de **SET NEW.*col_name* = *value*** requiert le droit de **UPDATE** sur la colonne.
- ✓ L'utilisation de **SET *value* = NEW.*col_name*** requiert le droit de **SELECT** sur la colonne.
- ✓ La commande **CREATE TRIGGER** requiert le droit de **SUPER**

III. LES TRIGGERS

III. 2. Création d’un trigger

Exemple :

```
Create Trigger ControleAge Before Insert On Etudiant For Each Row
Begin
        Declare n integer ;
        Set n = New.Age ;
        If n > 50 or n < 15 then
        Set New.Age = ‘Mon Age' ;
        End If ;
End ;
```

IV. LES ROUTINES

IV. 1. Qu’est-ce qu’une routine ?

- ✓ Les routines servent essentiellement à créer de petits programmes dont on a souvent besoin dans les tâches d’administration ;
- ✓ Ces programmes sont stockés dans la base et évitent les envois de messages entre les utilisateurs et le serveur ;
- ✓ Elles permettent donc de rendre le serveur plus performant ;
- ✓ Dans ce chapitre, nous parlerons des fonctions et des procédures.

IV. LES ROUTINES

IV. 2. Les procédures stockées

a. Qu’est-ce qu’une procedure stockée ?

- ✓ Une **procédure stockée** (ou ***stored procedure*** en anglais) est un ensemble d'instructions SQL pré-compilées, stockées sur le serveur, directement dans la base de données ;
- ✓ Elle peut être exécutée sur demande : lancée par un utilisateur, un administrateur DBA ou encore de façon automatisée par un déclencheur ;
- ✓ Les requêtes envoyées à un serveur SQL font l'objet d'une *analyse syntaxique* puis d'une *interprétation* avant d'être *exécutées*. Ces étapes sont très lourdes si l'on envoie plusieurs requêtes complexes ;

IV. LES ROUTINES

IV. 2. Les procédures stockées

a. Qu’est-ce qu’une procedure stockée ?

- ✓ Les instructions dans le corps d’une procédure ns sont envoyées qu'une seule fois sur le réseau puis analysée, interprétée et stockée sur le serveur sous forme exécutable **(pré-compilée)** ;
- ✓ Pour qu'elles soient exécutées, on fait appel à la procédure avec son nom ;
- ✓ On peut ainsi passer des paramètres à une procédure stockée lors de son appel, et recevoir le résultat de ses opérations comme celui de toute requête SQL.

IV. LES ROUTINES

IV. 2. Les procédures stockées

b. Création d'une procedure stockée

La syntaxe de création d'une procédure stockée est :

Create Procedure Nom_Procedure(IN|OUT|INOUT id_argument Type)

Begin

[Declaration (s) ;]

Instruction(s) ;

End ;

Remarque :

Une procédure est appelée avec la commande **Call**. Sa syntaxe est :

Call Nom_Procedure(Valeur_Arguments) ;

IV. LES ROUTINES

IV. 3. Les fonctions

a. Qu’est-ce qu’une fonction ?

- ✓ Une fonction est un ensemble d’instructions regroupées dans un module qui après exécution renvoie une valeur ;
- ✓ Elle peut prendre des arguments comme les procédures stockées ;
- ✓ La différence entre une fonction et une procédure est que la première renvoie un résultat, alors la dernière exécute une ou des opérations et ne renvoie pas de valeur.

IV. LES ROUTINES

IV. 3. Les fonctions

b. Création d'une fonction

La syntaxe de création de fonction est :

Create Function Nom_Fonction(arg_1 type, arg_n type) **Returns** Type_Retour

Begin

[Declaration (s) ;]

Instruction(s) ;

Return variable ;

End ;

Remarque : Une fonction est appelée comme suit :

Set @id_Var = Nom_Fonction(Arg_1, … Arg_n) ;

Select @id_Var ;

IV. LES ROUTINES

IV. 3. Les fonctions

c. Quelques fonctions utiles

Il existe des fonctions très utiles permettant de résoudre des problèmes rencontrés pendant la programmation sous MySQL :

i. **Concat**(c_1, c_2, …, c_n) : Elle permet de concaténer deux ou plusieurs chaines de caractères. Elle permet aussi de concaténer des chaines et des valeurs numériques.

Exemple :

```
Set @b = ‘Base de données relationnelle' ;
Set @x = Concat('On fait le cours de ', @b) ;
```

Alors @x contient : **On fait le cours de Base de données relationnelle**

IV. LES ROUTINES

IV. 3. Les fonctions

c. Quelques fonctions utiles

ii. **Substring**(var, debut, nb_caracteres) : Elle permet d'extraire une sous-chaine dans une chaine de caractères.

Remarque : Dans cette syntaxe, **var** est la variable contenant la chaine dans laquelle on veut extraire la sous-chaine, **debut** et la position du premier caractère à extraire et **nb_caracteres** est le nombre de caractères à extraire.

Exemple : Pour extraire Base de données dans la variable @x ci-dessus on fait :

Set @y = **Substring**(@x, 21, 15) ;

Alors la variable @y contient : **Base de données**

IV. LES ROUTINES

IV. 3. Les fonctions

c. Quelques fonctions utiles

iii. **Floor**(Réel) : Elle arrondit un nombre réel au plus grand entier inférieur à ce réel.

iv. **Ceil**(Réel) : Elle arrondit un nombre réel au plus petit entier supérieur à ce réel.

Exemple : Floor(20.5) = 20 ; Floor(-20.5) = -21 ; Ceil(20.5) = 21 ; Ceil(-20.5) = -20 ;

v. **Now() :** Cette fonction renvoie la date actuelle et l'heure sous le format : AAAA-MM-JJ HH:MM:SS.

Exemple : Select Now() ; /* Affiche **2022-12-08 11:04:52** */

IV. LES ROUTINES

IV. 3. Les fonctions

c. Quelques fonctions utiles

vi. **Date_Format**(Date, '%Partie') **:** Elle permet d'afficher la date sous le format voulu par l'utilisateur. Elle prend en paramètre une date et la partie de cette date à afficher.

- %y : Donne l'année sous le format AA ;
- %Y : Donne l'année sous le format AAAA ;
- %m : Donne le mois sous le format MM ;
- %M : Donne le mois sous le format texte en anglais (December) ;
- %d : Donne le jour sous le format JJ ;
- %D : Donne le jour sous le format JJth ou Jjrd, etc. ;
- %h : Donne l'heure sous le format 12h ;
- %H : Donne l'heure sous le format 24h ;
- %s et %S : Donnes les secondes sous le format SS ;
- %T : Donne l'heure au complet sous le format HH:MM:SS.

IV. LES ROUTINES

IV. 3. Les fonctions

Exemple :

```
Set @d = '2022-12-08' ;
Select Date_Format(@d, '%y') ; -- Affiche 22
Select Date_Format(@d, '%Y') ; -- Affiche 2022
Select Date_Format(@d, '%m') ; -- Affiche 12
Select Date_Format(@d, '%M') ; -- Affiche December
Select Date_Format(@d, '%d') ; -- Affiche 08
Select Date_Format(@d, '%D') ; -- Affiche 8th
Select Date_Format(Now(), '%T') ; -- Affiche 11:26:25th
```

V. LES CURSEURS

V. 1. Qu’est-ce qu’un curseurs

- ✓ Un curseur est un objet temporaire d’une base de données permettant de récupérer le résultat d’une requête qui renvoie une ou plusieurs enregistrements ;
- ✓ Il est sous la forme d’un tableau avec des lignes (enregistrements) et des colonnes (attributs) ;
- ✓ Dans un curseur, on peut parcourir les données et manipuler chaque enregistrement pour accomplir certaines tâches.

V. LES CURSEURS

V. 2. Utilisation d'un curseur

- ✓ **Déclaration :** Avant qu'un curseur ne puisse être utilisé, il doit être déclaré. Ce processus ne récupère pas les données, il définit simplement la requête qui lui fournit les données :

Declare Nom_Curseur **Cursor FOR Select …… ;**

- ✓ **Ouverture :** Après la déclaration, le curseur doit être ouvert pour être utilisé. Ce processus récupère les données fournies par la requête définie dans la déclaration du curseur ;

OPEN Nom_Curseur ;

- ✓ **Parcours :** Maintenant le curseur contient des données, des lignes individuelles peuvent être extraites selon les besoins en utilisant le mot clé ***FETCH*** et une boucle :

FETCH Nom_Curseur **INTO** v_1, v_2, …, v_n ;

- ✓ **Fermeture :** Après parcours du curseur, ce dernier doit être fermé :

V. LES CURSEURS

V. 3. Le gestionnaire Handler

- ✓ Lorsqu'on travaille avec des curseurs sous MySQL, on doit également déclarer un gestionnaire (**Handler**) pour gérer la situation où le curseur ne trouve aucune ligne ;
- ✓ À chaque fois que l'on appelle l'instruction **FETCH**, le curseur tente de lire la ligne suivante dans le curseur ;
- ✓ Lorsque le curseur atteint la fin du résultat, il ne pourra pas obtenir de données et une erreur est rencontrée ;
- ✓ Le gestionnaire est utilisé pour gérer cette erreur.

V. LES CURSEURS

V. 3. Le gestionnaire Handler

- ✓ Décration du gestionnaire

  Declare **Continue Handler** For **Not Found** Set Id_Var = 1 ;

- ✓ Id_Var est une variable qui permet de savoir si oui ou non le curseur a atteint la fin du résultat.
- ✓ La déclaration du gestionnaire doit apparaître après la déclaration de cette variable et du curseur.

V. LES CURSEURS

V. 4. Exemple

```
Declare n Varchar(15) ; Declare p Varchar(25) ; Declare a SmallInt ;
Declare vide Integer Default 0 ;
Declare Test_Curseur Cursor For Select Nom, Prenom, Age From Enseignant ;
Declare Continue Handler For Not Found Set vide = 1 ;
Open Test_Curseur ;
        Boucle_Loop : Loop
        Fetch Test_Curseur Into n, p, a ;
        If vide = 1 Then
        Leave boucle_Loop ;
        End If;
        Select n, p, a ;
        End Loop ;
Close Test_Curseur ;
```

VI. EXERCICE D'APPLICATION

Client (NumPermis, Nom, Prenom, Sexe, Adresse, Age, Telephone)

Voiture (Matricule, Marque, Version, Type, Couleur, Annee)

Louer (#Client, #Voiture, DateDebut, NbJour, PrixJour)

Fidelite (#Client, #Voiture, TotalNbJour, TotalPrix, Fidelite)

1. Créer une fonction qui renvoie le prix à payer pour un client qui loue une voiture à une date donnée ;
2. Créer une procédure qui affiche la liste des clients qui ont loué une voiture donnée pour une durée dépassant 7 jours ou "Cette voiture n'a jamais été louée plus de 7 jours" si personne n'a loué la voiture plus de 7 jours ;
3. Créer un trigger qui remplit automatiquement la table Fidelite à chaque fois qu'un client loue une voiture. Un client est fidèle à une voiture s'il l'a louée pour une durée cumulée dépassant 100 jours.