# Chapitre 2 : Introduction au PL/SQL sous MySQL

**Introduction :** Le langage SQL n'est pas un langage procédural. Il ne permet, donc pas de programmer l'exécution d'une action suivant un événement. Il est aussi impossible avec ce langage d'avoir des structures de contrôles ou des boucles pour répéter l'exécution d'une tâche.

Un langage procédural, repose sur des procédures, des fonctions ou des sous-routines prédéfinies et bien organisées dans l'architecture d'un programme en spécifiant 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. Des procédures ou fonctions sont implémentées sur les données et les variables pour effectuer une tâche. Ces procédures peuvent être appelées n'importe où entre la hiérarchie du programme et par d'autres procédures également.

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. La structure d'un code PL/SQL

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

a. L'entête ;
b. Les déclarations dans le corps du programme ;
c. Les instructions dans le corps du programme.

### I. 1. L'entête

L'entête d'un module PL/SQL contient le type de programme, son nom et éventuellement ses arguments (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. 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 handler, etc.
Devant chaque ligne de déclaration il y a le mot clé **Declare**.

**Exemple :**

**declare** id_variable type ;

### I. 3. Le corps du programme

Le corps d'un programme PL/SQL 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 PL-SQL

### 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**

  **Set** id_variable = valeur ;

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

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

**Exemple :**

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

### II. 2. Les opérateurs

Opérateurs ensemblistes : NOT IS NULL LIKE BETWEEN IN AND OR ANY

Opérateurs athmetiques : + - * / @ = <> <= >=

### II. 3. Les commentaires

```
-- Sur une seule ligne
/* Sur plusieurs
   ligne */
```

### II. 4. Les Instructions conditionnelles

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

**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 ;**

**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. 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** ;

**Exemple :**

```
Declare a integer ;
Declare b varchar(5) ;
Declare c integer Default 0 ;
Select Count(*) Into a From Etudiant Where VilleNaissance = 'Bignona' ;
If a = 5 then
    Set b = 'Il y a cinq étudiants nés à Bignona' ;
```

```
ElseIf a = 10 Then
        Set b = 'Il y a dix étudiants nés à Bignona' ;
End If ;
Select b ;
While a < 5 Do
        Set a = a + 1 ;
End While ;
Iterloop : Loop
        Set c = c + 5 ;
        If c >= 35 Then
                Leave iterloop ;
        End If ;
End Loop interloop ;
```

**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 de la en cours et de passer à une nouvelle

**g. 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 locale.

## III. Les Triggers ou déclencheurs

### III. 1. Qu'est-ce qu'un trigger ?

Un trigger est un objet de base de données associé à une table, qui s'active lorsqu'un événement particulier survient. Son exécution est déclenchée par la survenance d'une action bien déterminée. Il doit toujours faire référence à une table permanente.

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 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. 2. Création de triggers

La syntaxe de la création d'un trigger est :

**Create TRIGGER** Nom_Trigger Temps_Trigger Evenement_Trigger
**On** Nom_Tabble **For Each Row**
**Begin**
**[**Declaration (s) ;]
Instruction (s) ;
**End ;**

Un trigger est appelé de manière automatique selon son événement et son moment.

**Remarques :**

i. **Temps_Trigger** est soit **Before** soit **After**.
ii. **Evenement_Trigger** est soit **Insert** soit **Update** soit **Delete**

Dans la commande d'activation d'un déclencheur, on peut faire référence aux colonnes dans la table associée au déclencheur en utilisant les mots **OLD** et **NEW**.

**OLD.*col_name*** fait référence à une colonne d'une ligne existante avant sa modification ou son effacement.

**NEW.*col_name*** fait référence à une colonne d'une ligne après insertion ou modification.

**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**

**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 = Null ;
End ;
```

## IV. Les routines

Une routine est une fonction ou une procédure stockée. 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.

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. 1. Les procédures stockées

#### IV. 1. 1. Qu'est-ce qu'une procédure 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. Elles peuvent être exécutées sur demande : lancées par un utilisateur, un administrateur DBA ou encore de façon automatisée par un événement 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.

Les procédures stockées résolvent ce problème : une requête n'est envoyée qu'une seule fois sur le réseau puis analysée, interprétée et stockée sur le serveur **sous forme exécutable (pré-compilée)**. Pour qu'elle soit exécutée, le client n'a qu'à envoyer une requête comportant le nom de la procédure stockée.

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. 1. 2. Création de procédures stockées

La syntaxe de création d'une procédure stockée est :

**Create Procedure** Nom_Procedure(IN|OUT|INOUT id_variable Type)
**Begin**
**[**Declaration (s) ;]
Instruction(s) ;
**End ;**

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

**Call** Nom_Procedure(Valeur, Arguments) ;

### IV. 2. Les fonctions

#### IV. 2. 1. Création de Fonctions

La syntaxe de création de fonction est :

**Create Function** Nom_Fonction(Liste d'arguments avec leur type) **Returns** Type_Retour
**Begin**
[Declaration(s) ;]
Instruction(s) ;
Return Var_Retour ;
**End ;**

Une fonction est appelée comme suit :

**Set** @id_Var = Nom_Fonction(Valeur_Arguments) ;
**Select** @id_Var ;

#### IV. 2. 2. Quelques fonctions utiles

Il existe des fonctions très utiles permettant de résoudre des problèmes rencontrés durant l'administration d'une base de données :

**a. CONCAT**(liste de chaines à concaténer) : Elle permet de concaténer deux ou plusieurs chaines de caractères. Elle permet aussi de concaténer une chaine et un en entier.

**Exemple :**

Set @b = 'Samedi 08 Juin 2013' ;
Set @x = **concat**('On est le ', @b) ;
Alors @x contient : **On est le Samedi 08 Juin 2013**

**b. SUBSTRING**(var, debut, nb_caracteres) : Elle permet d'extraire une chaine d'une chaine de caractères. 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 le mot Samedi dans la variable @x ci-dessus on fait :

Set @y = **SUBSTRING**(@x, 11, 6) ;
Alors la variable @y contient : **Samedi**

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

**d. 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 ;

**e. Now() :** Cette fonction renvoie la date actuelle et l'heure sous le format :

AAAA-MM-JJ HH:MM:SS.

**Exemple :**

Select Now() ; /* Affiche **2013-06-12 11:51:26** */

**f. 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 lettre en anglais (January, February, etc.) ;

%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.

**Exemple :**

Select **Date_Format**('2022-02-18', '%y') ; -- Affiche **22**

Select **Date_Format**('2022-02-18', '%Y') ; -- Affiche **2022**

Select **Date_Format**('2022-02-18', '%m') ; -- Affiche **02**

Select **Date_Format**('2022-02-18', '%M') ; -- Affiche **February**

Select **Date_Format**('2022-02-18', '%d') ; -- Affiche **18**

Select **Date_Format**('2022-02-18', '%D') ; -- Affiche **18th**

A la date du **25 Novembre 2022**, on a :

Select Now() ; /* Affiche **2022-11-25 11:30:26** */

Select **Date_Format**(Now(), '%y') ; -- Affiche **22**

Select **Date_Format**(Now(), '%Y') ; -- Affiche **2022**

Select **Date_Format**(Now(), '%m') ; -- Affiche **11**

Select **Date_Format**(Now(), '%M') ; -- Affiche **November**

Select **Date_Format**(Now(), '%d') ; -- Affiche **25**

Select **Date_Format**(Now(), '%D') ; -- Affiche **25$^{th}$**

Select **Date_Format**(Now(), '%T') ; -- Affiche **11:30:26**

## V. Les curseurs

### V. 1. Qu'est-ce qu'un curseur ?

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 se présente 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. 2. Syntaxes de création et d'utilisation d'un curseur

L'utilisation de curseurs implique plusieurs étapes:

1. **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 …… ;**

2. **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 ;

3. **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** $v_1$, $v_2$, …, $v_n$ ;

4. **Fermeture :** Après parcours du curseur, ce dernier doit être fermé :

   **CLOSE** Nom_Curseur ;

### V. 3. Le gestionnaire HANDLER

Lorsqu'on travaille avec des curseurs sous MySQL, on doit également déclarer un gestionnaire (**HANDLER**) **NOT FOUND** 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.

Syntaxe de déclaration 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 dans les procédures stockées ou fonctions.

**Exemple :**

```
Declare n Varchar(15) ;
Declare p Varchar(25) ;
Declare a SmallInt ;
Declare C1 Cursor For Select Nom, Prenom, Age From Enseignant ;
Declare vide Integer Default 0 ;
Declare Continue Handler For Not Found Set vide = 1 ;
Open C1 ;
    Boucle_Loop : Loop
        Fetch C1 Into n, p, a ;
        If vide = 1 Then
                Leave boucle_Loop ;
        End If;
        Select n, p, a ;
    End Loop ;
Close C1 ;
```

## VI. Exercice d'application

**Client** (NumPermis, Nom, Prenom, Sexe, Adresse, Age, Telephone)
**Voiture** (Matricule, Marque, Version, Type, Couleur, Annee)
**Louer** (#Client, #Voiture, Date, NbJour, PrixJour)
**Fidelite** (#Client, #Voiture, TotalNbJour, TotalPrix, Fidele)

1. Créer une fonction qui renvoie le prix à payer pour un nombre de jours et un prix journalier donné ;
2. Créer une procédure qui affiche la liste des clients qui ont loué la voiture de matricule ZG 0000 P pour une durée dépassant une semaine (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.