Utilisation d'une base MySQL

L'utilisation d'une base MySQL est possible à l'aide d'un ensemble de classes définies dans l'espace de nom **MySql.Data.MySqlClient;**

Cette espace de nom n'est cependant pas disponible par défaut sous Visual Studio, il faut l'installer (voir documentation)

3 classes principales sont disponibles :

Classe	Rôle
MySqlConnection	Elle permet de spécifier les caractéristiques d'une connexion et d'ouvrir une connexion vers une base de données.
MySqlCommand	Elle permet d'exécuter des requêtes SQL sur la base de données
MySqlDataReader	Elle permet de parcourir le résultat d'une requête SQL retournant plusieurs lignes

Pour pouvoir accéder aux méthodes de ces classes il faut ajouter la directive : **using MySql.Data.MySqlClient;**

1. La classe MySqlConnection

Constructeur	Paramètre
MySql**Connection**(chaine)	Chaîne de connexion (voir les chaînes de connexion)

Méthode	Rôle
Open()	Ouverture de la connexion (exception générée en cas d'erreur)
Close()	Fermeture explicite de la convexion

Propriété	Rôle
ConnectionString	Contient la chaîne de connexion

2. La classe MySqlCommand

Constructeur	Paramètre
MySqlCommand()	Constructeur par défaut
MySqlCommand(maRequete, maConnexion)	La requête SQL et l'objet MySqlconnection

Méthode	Rôle
ExecuteNonQuery()	Exécute une requête de mise à jour (Insert, Update, delete). Retourne le nombre de lignes affectées.
ExecuteScalar()	Exécute une requête ne renvoyant qu'une valeur. Retourne la valeur sous la forme d'un objet qu'il faudra donc transformer.
ExecuteReader()	Exécute une requête renvoyant plusieurs lignes. Retourne un objet MySqlDataReader qu'il faut ensuite parcourir.

Propriété	Rôle
Connection	Obtient ou définit l'objet MySqlConnection.
CommandText	Texte de la requête SQL ou nom d'une procédure stockée ou nom d'une table.
CommandType	Indique le type de contenu de la propriété CommandText : System.Data.CommandType.Text ➔ commande SQL System.Data.CommandType.StoredProcedure ➔ Nom d'une procédure
Parameters	Dictionnaire des paramètres utilisés dans la requête SQL ou la procédure

Méthode sur Parameters	Rôle
AddWithValue("nom", valeur)	Ajoute un paramètre identifié par @nom et initialisé avec valeur
Add("nom", type, [size])	Ajout un paramètre en précisant son type (SqlDbType)
Clear()	Supprime tous les paramètres de la collection.

Propriété sur un paramètre	Rôle
SqlDbType	Type du paramètre (`System.Data.SqlDbType.VarChar`, …)
Direction	Indique System.Data.ParameterDirection.Input (Output, inputOutput, `ReturnValue`)
Value	Retourne la valeur du paramètre sous la forme d'un objet qu'il faudra donc convertir
Size	Taille du paramètre (à préciser pour un type varchar si le paramètre est en sortie)

3. La classe MySQLDataReader

Elle permet de parcourir le résultat d'une requête retournant plusieurs lignes (**ExecuteReader()**)

Propriété	Rôle
FieldCount	Nombre de colonnes dans la ligne en cours
Item ou []	Valeur d'une colonne dans son format natif L'indice peut correspondre au numéro de la colonne en commençant par 0 ou au nom du champ dans le jeu d'enregistrements.

Méthode	Rôle
Read()	Avance jusqu'à l'enregistrement suivant. Le premier Read se place sur le premier enregistrement. Renvoie faux s'il n'y a plus d'enregistrement
Close()	Fermeture du jeu d'enregistrement
GetString(n)	Retourne la valeur de la colonne n dans un objet string
GetDateTime(n)	Retourne la valeur de la colonne n dans un objket DateTime
GetByte(n)	Retourne la valeur de la colonne n dans le format Byte (octet)
GetInt16(n)	Retourne la valeur de la colonne n dans le format Int16 (entier court)
ou encore	GetInt32(n), GetDouble(n), GetDecimal(n), GetChar(n)
IsDBNull(n)	Retourne vrai si la colonne n contient la valeur null. Cette méthode doit être utilisée avant les méthodes Get pour éviter toute erreur

Le paramètre n correspond au numéro d'ordre du champ dans la clause Select. La numérotation commence à 0.

4. La connexion

4.1. Connexion à l'aide d'une chaine de connexion

```
string chaineConnexion = "Data Source=localhost;Database=secolog; User Id=root; Password=";
MySqlConnection  cnx = new MySqlConnection(chaineConnexion);
cnx.Open();
```

Si le serveur n'est pas lancé ou n'existe pas on obtient le message d'erreur suivant :
Erreur: Unable to connect to any of the specified MySQL hosts.

4.2. Connexion à l'aide du fichier de configuration App.config

Les paramètres d'accès à la base de données, peuvent être stockés dans la section <connectionStrings> du fichier de configuration App.config

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="sgbd"
         providerName="MySql"
        connectionString=" Data Source=localhost;Database=secolog; User Id=root; Password="/>
  </connectionStrings>
</configuration>
```

Pour pouvoir accéder à ce fichier il faut :

- Au niveau du projet ajouter la référence System.configuration
  Menu Projet – Ajouter une référence – Onglet Assemblys - Framework – cocher la case sur la ligne System.configuration
- Au niveau du fichier '.cs', ajouter la référence : using System.Configuration;

L'ouverture d'une connexion s'effectue alors de la façon suivante :

```
ConnectionStringSettings uneChaine = ConfigurationManager.ConnectionStrings["sgbd"];
MySqlConnection cnx = new MySqlConnection(uneChaine.ConnectionString);
cnx.Open();
```

Ces trois instructions doivent être de préférence centralisées dans une classe Passerelle en charge de toutes les interactions avec la base de données(Cf 4.)

5. Exécuter une requête.

Il faut commencer par créer un objet MySqlCommand en précisant la requête SQL et l'objet connexion utilisé.

5.1. Instanciation d'un objet MySqlCommand

Elle s'effectue en appelant le constructeur de la classe MySqlCommand qui possède plusieurs surcharges.

Version 1 : Les paramètres (requête et objet connexion) sont transmis en paramètre au constructeur.

```
string sql = "Select …";
MySqlCommand  cmd = new MySqlCommand(sql, cnx);
cmd.CommandType = System.Data.CommandType.Text;
```

il n'exisste pas de constructeur permettant d'initialiser la propriété CommandType mais Text étant la valeur par défaut la dernière ligne peut être supprimée

Inconvénient : il faut connaitre l'ordre des paramètres à fournir

Version 2 : Les paramètres sont renseignés après l'instanciation par le constructeur par défaut.

```
MySqlCommand  cmd = new MySqlCommand();
cmd.CommandType = System.Data.CommandType.Text;
cmd.CommandText = sql;
cmd.Connection = cnx;
```

Version 3 : Les paramètres sont renseignés dans le corps après l'instanciation par le constructeur par défaut.

```
MySqlCommand  cmd = new MySqlCommand() {
    CommandType = System.Data.CommandType.Text;
    CommandText = sql;
    cmd.Connection = cnx;
}
```

Cette solution peut aussi utiliser les autres constructeurs de la classe.

Il est possible de réutiliser le même objet MySqlCommand pour lancer une nouvelle requête, il suffit pour cela de modifier sa propriété CommandText et éventuellement sa propriété *CommandType.*

Cependant attention à réinitialiser si nécessaire la collection des paramètres de l'objet si elle a été précédemment alimentée et à bien vérifier que tous les curseurs utilisés ont bien été fermés :

MySql.Data.MySqlClient.MySqlException : 'There is already an open DataReader associated with this Connection which must be

5.2. Lancement d'une requête retournant une seule ligne possédant une seule valeur

La méthode ExecuteScalar() répond à ce type de demande. Elle retourne un objet de type Object contenant le résultat. Il faut bien sur appliquer un cast de type string pour en récupérer la valeur

```
string sql = "Select nom from client where id = 1;";
cmd.CommandText = sql;
string unNom = (string) cmd.ExecuteScalar();
```

Si la requête est susceptible de ne retourner aucune valeur, il faut stocker le résultat dans une variable de type object est tester si sa valeur est null

```
object valeur = cmd.ExecuteScalar();
if (valeur != null) {…}    ou if(!valeur is null)
```

5.3. Lancement d'une requête retournant plusieurs lignes

Il faut déclarer un objet de la classe MySqlDataReader pour stocker les lignes retournées par la méthode ExecuteReader();

Le parcours des lignes s'effectue à l'aide de la méthode Read de l'objet MySqlDataReader.
Pour récupérer les valeurs de chaque champ, il faut procéder à des conversions. Plusieurs solutions sont possibles.

On peut utiliser les méthodes 'Get' de l'objet MySqlDataReader : GetString(n), GetDateTime(n)

On peut aussi si la conversion peut être implicite réaliser un cast : (string)

On peut aussi utiliser la redéfinition de la méthode ToString()

```
string sql = "Select id, nom, prenom, dateNaissance from Eleve ";
MySqlCommand cmd = new MySqlCommand(sql, cnx);
MySqlDataReader curseur = cmd.ExecuteReader();
while (curseur.Read()) {
    int idEleve = curseur.GetInt32(0);
    string nomEleve = curseur["nom"].ToString();
    string prenomEleve = (string) curseur["prenom"];
     DateTime date = curseur.GetDateTime(3);
}
curseur.Close();
```

On peut utiliser un type implicite pour la variable curseur var curseur = cmd.ExecuteReader()
C'est un avantage si l'on doit changer de SGBDr

Si un champ de la requête contient la valeur null, elle ne peut être stockée dans une variable d'un type string ou int. Il faut utiliser la méthode IsDbNull(n) pour tester la valeur du champ

Exemple : le troisième champ retourné est de type décimal acceptant la valeur Null

```
taux = (curseur.IsDBNull(2) ? 0 : unCurseur.GetDecimal(2));
```

Si la requête contient des paramètres, il faut utiliser la propriété Parameters de l'objet MySqlCommand
Cette propriété représente un dictionnaire de type clé, valeur qui va stocker la valeur de chaque paramètre.
Pour ajouter un paramètre dans la propriété Parameters, il faut utiliser la méthode AddWithValue("nom", valeur)

Dans la requête SQL les paramètres peuvent être nommés ou non (utilisation de ?) mais le mélange n'est pas possible.

```
List<Produit> lesProduits = new List<Produit>();
sql = "Select id, nom from produit where idCategorie = @idCategorie";
cmd = new MySqlCommand(sql, cnx);
cmd.Parameters.AddWithValue("idCategorie", idCategorie);
var curseur = cmd.ExecuteReader();
while (curseur.Read()) {
        int id = (int) curseur["id"];
        string nom = (string ) curseur["nom"];
        lesProduits.Add(new Produit(id, nom));
}
curseur.Close();
```

5.4. Lancement d'une requête sans paramètre réalisant une opération de mise à jour

La méthode ExecuteNonQuery() répond à ce type de demande.

```
string sql = "delete from produit where id = 1;";
MySqlCommand  cmd = new MySqlCommand(sql, cnx);
cmd.ExecuteNonQuery();
```

5.5. Lancement d'une requête paramétrée réalisant une opération de mise à jour

```
string sql = "Update produit set prix = @prix where id = @id";
MySqlCommand  cmd = new MySqlCommand(sql, cnx);
cmd.Parameters.AddWithValue("prix", prix);
cmd.Parameters.AddWithValue("id", code);
cmd.ExecuteNonQuery();
```

Si on doit relancer la requête avec d'autres valeurs pour les paramètres :

```
cmd.Parameters["prix"].Value = 110;
cmd.Parameters["id"].Value = 5;
```

Dans ce cas on doit souvent déclarer les paramètres sans leur attribuer de valeur et ensuite dans une structure itérative initialiser la valeur de chaque paramètre.

La méthode Add("nomParamètre", MySqlDbType.*Int32*) de la propriété Parameters permet de déclarer un paramètre.
Valeur possible pour le second paramètre :
https://dev.mysql.com/doc/dev/connector-net/8.0/html/T_MySql_Data_MySqlClient_MySqlDbType.htm
Pour la valeur MySqlDbType.VarChar un troisième paramètre précisant la taille est nécessaire

Exemple : Ajouter un ensemble de produit contenu dans une collection de Produit lesProduits

```
sql = "Insert into Produit (id, designation) values (@id, @designation)";
cmd = new MySqlCommand(sql, cnx);
cmd.Parameters.Add("id", MySqlType.Int32);
cmd.Parameters.Add("designation", MySqlType.VarChar, 50);
foreach (Produit unProduit in lesProduits) {
        cmd.Parameters["reference"].Value = unProduit.Reference;
        cmd.Parameters["designation"].Value = unProduit.Designation;
        cmd.ExecuteNonQuery();
}
```

5.6. Récupération de la valeur d'un champ de type compteur

```
string sql = "Select @@identity";
MySqlCommand cmd = new MySqlCommand(sql, cnx);
int numStagiaire = (int) cmd.ExecuteScalar();
```

Requête spécifique pour MySQL

```
cmd.CommandText = "Select last_insert_id();";
object valeur = cmd.ExecuteScalar();
if (valeur != null) { numstagiaire = Int32.Parse(valeur.ToString()); }
```

6. Utilisation d'une procédure ou fonction stockée

Une procédure ou fonction stockée représente une procédure ou une fonction définie sur le SGBDr
L'ensemble représente une interface entre une application C# et le serveur MySQL.
Le programmeur n'a plus à connaitre la structure de la base utilisée (tables) ni même la syntaxe du langage SQL.

Exemple :

```
create procedure getLesVisites(idVisiteur varchar(3))
   Select id, date, heure, bilan, …  from visite
   Where Visite.idVisiteur = idVisiteur;
// attention de bien lever tout ambiguïté entre les paramètres et le nom des attributs

create function compteParticipant(f char(4), s integer) returns int DETERMINISTIC
  Begin
    declare nb int;
    Select count(*) into nb from participer where idFormation = f and numSession = s;
    return nb;
  end
```

L'appel d'un procédure s'effectue en paramétrant les propriétés de l'objet MySqlCommand cmd

```
MySqlCommand cmd = new MySqlCommand() {
        Connection = cnx,
        CommandText = "getLesVisiteurs",
        CommandType = CommandType.StoredProcedure,
};
```

Nécessite : using System.Data;

Si la procédure nécessite des paramètres on les définit comme pour une requête paramétrée en précisant leur direction (pour les paramètres en sortie).
Si la procédure retourne un jeu d'enregistrements on effectue la lecture au moyen d'un curseur.

```
cmd.Parameters.AddWithValue("idVisiteur", id);
curseur = cmd.ExecuteReader();
while (unCurseur.Read())    {
   string nom = unCurseur['nom'].ToString()
   …
}
curseur.Close();
```

Exécution d'une fonction stockée (solution de contournement)

```
MySqlCommand cmd = new MySqlCommand() {
        Connection = cnx,
        CommandText = " Select compteParticipant(id, numSession)",
        CommandType = CommandType.Text,
};
cmd.Parameters.AddWithValue("id", idFormation);
cmd.Parameters.AddWithValue("numSession", numSession);
int nb = (int)cmd.ExecuteScalar();
```

Exécution d'une procédure possédant un paramètre en sortie

C'est le cas d'une procédure qui ajoute une occurrence dans une table et qui doit transmettre la valeur de l'identifiant de type compteur.

```
MySqlCommand cmd = new MySqlCommand()  {
    Connection = cnx,
    CommandText = "ajouterRendezVous",
    CommandType = CommandType.StoredProcedure,
 };
 cmd.Parameters.AddWithValue("idVisiteur", idVisiteur);
 cmd.Parameters.AddWithValue("idPraticien", idPraticien);
 cmd.Parameters.AddWithValue("idMotif", idMotif);
 cmd.Parameters.AddWithValue("dateEtHeure", uneDate);
 cmd.Parameters.Add("idVisite", MySqlDbType.Int32);
 cmd.Parameters["idVisite"].Direction = ParameterDirection.Output;
 cmd.ExecuteNonQuery();
 int idVisite = (Int32)cmd.Parameters["idVisite"].Value;
```

7. Utilisation d'une classe Passerelle

Il est fortement recommandé de centraliser l'ensemble des opérations concernant la base de données dans une classe Passerelle.
Cette classe contient un membre statique cnx contenant l'objet MySQLConnexion instancié avec la chaine de connexion contenu dans le fichier de configuration.
Cette classe contient des méthodes publiques assurant l'interaction avec la base de données.
On trouve en générale une méthode chargerDonnee() qui permet d'alimenter les objets de l'application et des méthodes pour ajouter, supprimer, modifier les données.

```
using System;
using MySql.Data.MySqlClient;
using System.Configuration;

public class Passerelle  {
   static private ConnectionStringSettings uneChaine = ConfigurationManager.ConnectionStrings["sgbd"];
   static private MySqlConnection cnx = new MySqlConnection(uneChaine.ConnectionString);

   public static Donnees chargerDonnees()   {
         cnx.Open();
         String sql = "Select id, nom from client";
         MySqlCommand cmd = new MySqlCommand(sql, cnx);
         MySqlDataReader curseur = cmd.ExecuteReader();
         while (curseur.Read()) {
            int id = (int) curseur["id"];
            string nom = curseur["nom"].ToString();
            Globale.LesClients.Add(new Client(id, nom));
         }
         curseur.Close();
         ….
         cnx.Close();
         return mesDonnees;
   }

   public static void ajouterParticipant(string idFormation, int numSession, int idSalarie) {
         cnx.Open();
         string sql = @"
               insert into participer (idFormation, numSession,idStagiaire)
               values(@idFormation, @numSession, @idStagiaire);";
         MySqlCommand cmd = new MySqlCommand(sql, cnx);
         cmd.Parameters.AddWithValue("idFormation", idFormation);
         cmd.Parameters.AddWithValue("numSession", numSession);
         cmd.Parameters.AddWithValue("idStagiaire", idSalarie);
         cmd.ExecuteNonQuery();
         cnx.Close();
   }
   …
}
```

Globale est une classe contenant des membres 'static' 'public' de type collection ou dictionnaire stockant toutes les données nécessaires à l'application.
Il est possible d'utiliser MySqlCommand cmd = new MySqlCommand() { …} pour instancier l'objet cmd
Il est recommander de remplacer les requêtes par des procédures stockées qui sont toujours plus rapides et qui assurent une indépendance entre l'application et la base de données.

8. Mise en place d'une transaction

Une transaction permet d'enregistrer un ensemble de requêtes de mise à jour comme s'il s'agissait d'une seule et même requête.
Si l'une d'entre elles échoue, aucune n'est réalisée
Plus exactement *celles qui ont déjà été exécutées sont supprimées ce qui n'est pas tout à fait la même chose; En effet si la transaction contient une instruction insert sur une table dont la clé primaire est de type compteur, et que cette instruction est exécutée avant que la transaction soit annulée, la valeur du compteur attribué ne sera pas réutilisable pour un prochain insert.*

Cela est intéressant lorsqu'un objet du système est représenté dans la base de données par plusieurs table.
Par exemple : une vente : table vente + table detailvente (les produits contenu dans la vente)

Un transaction est démarrée avec la méthode **BeginTransaction()** de l'objet Connexion
Elle est validée avec la méthode avec la méthode **Commit()**
Elle peut être annulée à tout moment avant la validation par la méthode **Rollback()**
Les commandes lancées dans la transaction doivent être attachées à cette transaction.

Exemple

```
cnx.Open();
MySqlTransaction uneTransaction = cnx.BeginTransaction();
MySqlCommand cmd = new MySqlCommand() {
     Connection = cnx,
     CommandText = "…",
     CommandType = CommandType.StoredProcedure,
     Transaction =  uneTransaction
};
 …
try   {
     cmd.ExecuteNonQuery();
      cmd.CommandText = "…",
      cmd.ExecuteNonQuery();
      …
      uneTransaction.Commit();
       cnx.Close();
} catch (System.Data.SqlClient.SqlException e)  {
         uneTransaction.Rollback();
          cnx.Close();
}
```