TP5 SLAM2 : GESTION IMMOBILIERE
C# ET ACCES BD

Objectifs :
- C# et Accès à une BD : connexion, récupération des données (SELECT), insertion de données (INSERT)
- Organiser son développement selon le principe de Modèle-Vue-Contrôleur (MVC)

Document joint : Suite du TP4

Durée : 2 séances (19 et 26 Mars 2021)

Objectifs de l'application

Suite du TP4 : il vous est désormais possible d'enregistrer dans une collection des biens immobiliers, de les modifier ou de les supprimer.
L'objectif de ce TP est de permettre un enregistrement dans une base de données afin de les lister et d'en insérer des nouveaux.

PARTIE 1 : MISE EN PLACE DE LA BASE DE DONNEES `SLAM2_BD_IMMOBILIER` SOUS MYSQL

Pour enregistrer les biens immobiliers, on dispose donc de la base de données suivante `SLAM2_BD_IMMOBILIER` :

BIEN (idBien, adresseBien, villeBien, surface, nbPieces, nbChambres, nbSallesEau, prix, typeBien)

Légende : clé primaire, clé étrangère#
La clé primaire idBien sera en auto-increment.

- **Mettre en place la BD répondant à cette problématique.**

- **Insérer quelques tuples exemples** pour tester votre application.

PARTIE 2 : ENVIRONNEMENT DE DEVELOPPEMENT

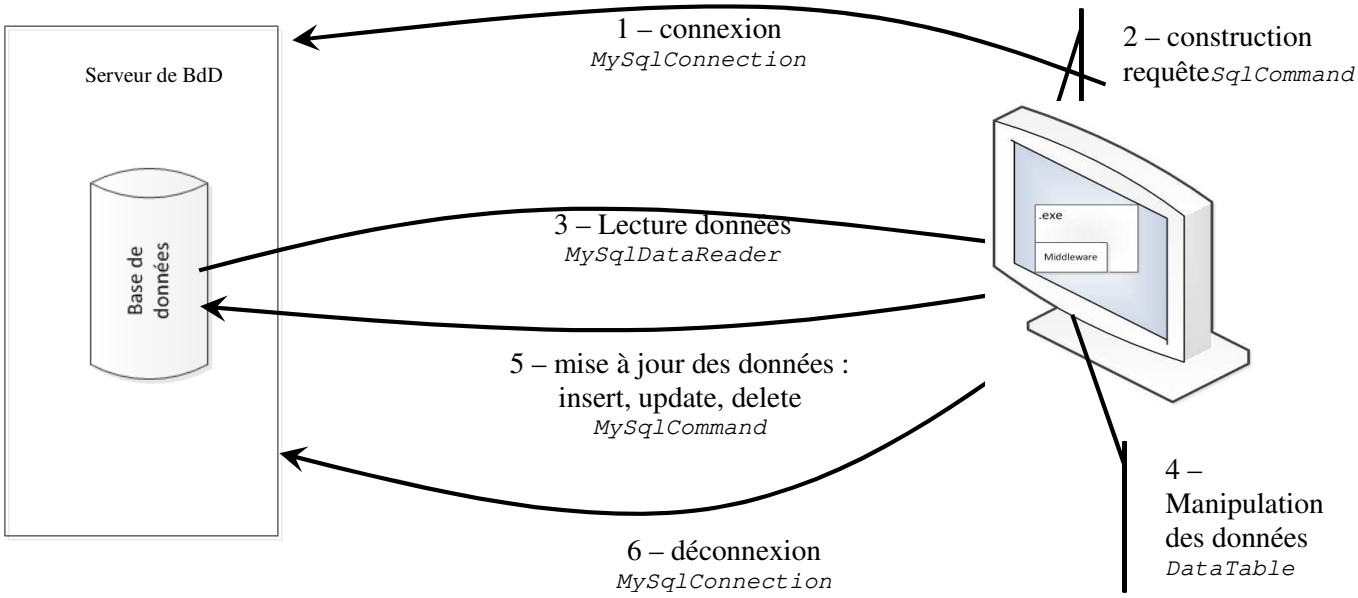
ADO.NET est un ensemble de classes qui proposent les services d'accès aux données pour les programmeurs utilisant le Framework .NET.

MySQL n'est pas supporté nativement dans l'environnement .NET. Mais, pour remédier à ce problème, vous pouvez télécharger différents connecteurs qui vous permettront d'intégrer MySQL dans votre IDE (un connecteur MYSQL est mis à disposition dans le dossier du TP sur MOODLE : mysql-connector-net-8.0.23)
Ou vous pourrez télécharger gratuitement à l'adresse suivante: http://dev.mysql.com/downloads/connector/net/,
lancer l'installation en double-cliquant sur le fichier.msi.

Une fois que tout est ouvert, dans l'explorateur de solution, faites un clic droit sur **"Références"** et choisissez **"Ajouter une référence"**.
Si l'installation du provider s'est bien passée, vous devriez voir une nouvelle référence à ajouter dans vos projets (MySQL.Data).

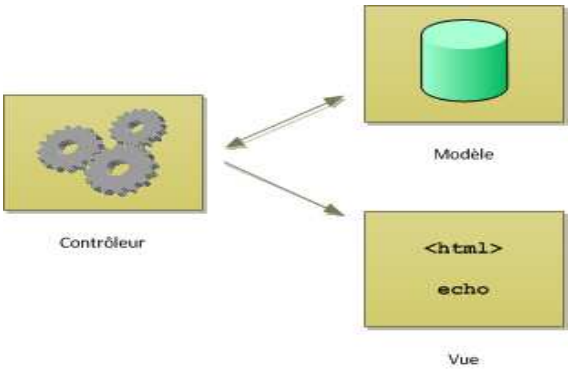
Il faudra bien ajouter les bibliothèques suivantes dès que vous utiliserez le lien avec la BD :

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



Remarque :L'accès aux données est identique quelque soit la base de données choisie, seule la chaîne de connexion change pour s'adapter au moteur choisi. Ici, je vous propose de tester avec une base de données sous MySQL, à vous d'adapter la connexion si vous souhaitez une autre BD.

MODELE VUE CONTROLEUR :



PARTIE 3 : CREATION DE LA CLASSE MODELE

- Créez une **nouvelle classe publique** dans le projet du nom « Modele.cs » qui va permettre de différencier la gestion graphique (la vue) à la gestion de l'accès aux données (le modèle).

Le *Modèle* représente le traitement des données, les interactions avec la base de données, etc. Il décrit les données manipulées par l'application et définit les méthodes d'accès.

- Créer un constructeur à vide : `public Modele() { }`

- Ajoutez dans les bibliothèques :

```
using MySql.Data.MySqlClient;
using System.Data;
using System.Windows.Forms;
```

- Ajoutez dans les variables privées de la form :

```
private MySqlConnection myConnection; // objet de connexion
private bool connopen = false; // test si la connexion est faite
private bool chargement=false; // test si le chargement d'une requête est fait
```

- Ajoutez les **accesseurs lecture** de connopen, chargement.
- Ajoutez les **2 méthodes** qui permettent de se connecter et de se déconnecter :

```
        /// <summary>
        /// Méthode pour se connecter à la BD
        /// </summary>
        public void seconnecter()
        {
            // paramètres de connexion à modifier selon sa BD et son serveur de BD
            string myConnectionString = "Database=SLAM2_BD_IMMBOLIER;Data Source=localhost;User
Id=root; Password=;";
            myConnection = new MySqlConnection(myConnectionString);
            try // tentative
            {
                myConnection.Open();
                connopen = true;
            }
            catch (Exception err)// gestion des erreurs
            {
                MessageBox.Show("Erreur ouverture bdd : " + err, "PBS connection",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                connopen = false;
            }
        }

        /// <summary>
        /// Méthode pour se déconnecter de la BD
        /// </summary>
        public void sedeconnecter()
        {
            if (!connopen)
                return;
            try
            {
                myConnection.Close();
                myConnection.Dispose();
                connopen = false;
            }
            catch (Exception err)
            {
                MessageBox.Show("Erreur fermeture bdd : " + err, "PBS deconnection",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
```

Nous compléterons cette classe modele dans la suite du TP selon les besoins d'utilisation des données.

PARTIE 4 : CREATION DE LA CLASSE CONTROLEUR

- Créez une nouvelle **classe statique** dans le projet du nom « controleur.cs » qui va permettre de faire les traitements entre le modèle (accès à la BD) et la vue (toutes vos forms).
  Une classe statique permet d'être utilisée sans instancier d'objet de cette classe.
  Les méthodes sont directement accessibles par la classe.

```
public static class Controleur
    {
```

- Déclarer une variable static vmodele de type modele : `private static Modele vmodele;`
- Déclarer une méthode public static void init() qui permet d' instancier vmodele (pas de la déclarer, juste l'instancier).

- Ajouter l'assesseur de vmodele

Nous compléterons cette classe controleur dans la suite du TP selon les besoins de traitements.

PARTIE 5 : CONNEXION A LA BD

Nous allons travailler en mode connecté, donc nous allons nous connecter au chargement de la feuille principale (sur l'évènement Load) et nous déconnecter en quittant l'application.

- **Appel de la méthode de connexion** sur l'évènement Load de la form principale :

- Appeler la méthode init() du controleur (pour instancier la propriété Vmodele du modele). Pas besoin d'instancier d'objet de la classe controleur pour l'utiliser ici puisque la classe est statique : `Controleur.init();`
- Appeler la méthode seconnecter de la classe modele via la propriété Vmodele du controleur : `Controleur.Vmodele.seconnecter();`
- Si la connexion n'a pas eu lieu (propriété Connopen de modele à false) : indiquer un messageBox d'erreur de connexion
- Sinon (connexion ok) : indiquer un message de connexion OK.

- **Et la déconnnexion** :

Puisque nous sommes en mode connecté en permanence, la déconnexion aura lieu au moment de quitter l'application :
Gérez l'événement FormClosed pour la forme principale dans lequel vous appellerez la méthode sedeconnecter() à partir de la classe modele :

```
private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
    Controleur.Vmodele.sedeconnecter();
}
```

- **Tester et vérifier que la connexion et la déconnesion se passe bien.**

PARTIE 6 : GESTION DES DONNEES dans le Modele

Toutes les données seront chargées dans des dataTable selon les besoins pour la consultation.
On utilise les objets :

- **myConnection :** pour la connexion à la BD (déjà utilisé)
- **MySqlCommand** pour créer les requêtes (select, insert, update, delete)
- **dataTable** qui va récupérer les résultats de l'exécution de notre commande SQL
- **MySqlDataReader** pour lire le dataTable

Vous déclarerez autant de dataTable que besoin (ou créer des collections).

- Déclarer dans la classe modele :

```
private DataTable dT1 = new DataTable();
```

- Créez les accesseurs pour les DataTable

- Déclarer dans Modele la méthode générique `charger()` suivante qui va remplir le dataTable.

BIEN LA COMPRENDRE POUR POUVOIR LA REUTILISER

```
        /// <summary>
        /// Méthode générique pour charger les données issues d'une requête dans un dataTable
        /// </summary>
        /// <param name="requete"></param>
        /// <param name="DT"></param>
        public void charger(string requete, DataTable DT)
        {
            MySqlCommand command = myConnection.CreateCommand();
            MySqlDataReader reader;

            try
            {
                command.CommandText = requete;
                reader = command.ExecuteReader();
                DT.Load(reader);
                chargement = true ;
            }
            catch (Exception err)
            {
                MessageBox.Show("Erreur chargement dataTable: " + err, "PBS table",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                chargement = false;
            }
        }
```

Cette méthode permet de charger en mémoire dans un DataTable les résultats d'une requête.
Il faudra appeler cette méthode avec la bonne requête, le bon dataTable selon la table utilisée.

Pour rassembler tous nos chargements, nous allons créer une méthode qui va rassembler tous nos appels :

```
        public void charger_donnees(string table)
        {
            if (table == "bien") charger("select * from bien;", dT1);
        }
```

PARTIE 7 : LECTURE DES DONNEES DANS LA VUE (VIA UN SELECT)

Une fois les données chargées dans les DataTable, vous allez pouvoir les utiliser pour les consulter, les modifier, en ajouter ou en supprimer. Ici, nos biens sont listés dans une liste déroulante de type comboBox à partir d'une collection de biens, nous allons donc récupérer les données de la BD pour les copier dans la collection.

- Pour les consulter et remplir la collection lesBiens :

Sur l'évènement Load de la form, vous avez effectué la connexion à la BD. Si la connexion s'est bien passée, nous allons parcourir les données de la BD remplies dans le dataTable dt1 pour les copier dans votre collection lesBiens.

// déjà fait en partie : à compléter dans le ELSE

```
        private void FormPrincipale_Load(object sender, EventArgs e)
        {
            Controleur.init();
            Controleur.Vmodele.seconnecter();
            if (Controleur.Vmodele.Connopen == false)
            {
                MessageBox.Show("Erreur dans la connexion");
            }
            else
            {
                MessageBox.Show("BD connectée");
                Controleur.Vmodele.charger_donnees("bien");
                // si le chargement s'est bien passé
                if (Controleur.Vmodele.Chargement)
                {
                    DataTable dt = Controleur.Vmodele.DT1;
```

```
                    // remplissage de la collection à partir du dataTable chargé
                    // on parcourt tout le dataTable
                    for (int i = 0; i < Controleur.Vmodele.DT1.Rows.Count; i++)
                    {
                        // on ajoute chaque tupe du dataTable dans la collection
                        lesBiens.Add(new Bien(Convert.ToInt32(dt.Rows[i]["typeBien"]),
dt.Rows[i]["villeBien"].ToString(), dt.Rows[i]["adresseBien"].ToString(),
Convert.ToInt32(dt.Rows[i]["surface"]), Convert.ToInt32(dt.Rows[i]["nbPieces"]),
Convert.ToInt32(dt.Rows[i]["nbChambres"]), Convert.ToInt32(dt.Rows[i]["nbSallesEau"]),
Convert.ToInt32(dt.Rows[i]["prix"])));
                    }
                    chargerCombo(); // méthode à ajouter plus bas
                }
            }
```

- votre collection lesBiens est désormais remplie avec les données de la BD, faire une méthode qui reprend votre code pour remplir la comboBox avec les biens de la collection :

```
        private void chargerCombo()
        {
            cbListeBiens.Items.Clear();
            foreach (Bien B in lesBiens)
            {
                cbListeBiens.Items.Add(B.RetourneTypeBien() + " - " + B.Ville + " " +
B.NbPieces + " pièces - " + B.Prix + "€");
            }
        }
```

- Cette méthode doit être rappelée dès qu'une modification a lieu dans la collection : à appeler à la place de la méthode Load sur les boutons AJOUTER, SUPPRIMER et MODIFIER.

- **Tester et vérifier que le chargement des données dans la liste déroulante comboBox se passe bien.**

Nous verrons par la suite d'autres méthodes pour lister toutes les données, dans une dataGridView par exemple.

PARTIE 8 : AJOUT DES DONNEES (VIA UN INSERT)

Pour ajouter des données dans la BD, on va utiliser le principe des requêtes préparées et récupérer les champs via l'interface.

- Dans la classe Modele, on va donc créer une nouvelle méthode pour préparer l'insertion :

```
        /// <summary>
        /// Méthode qui permet d'ajouter un bien avec l'ensemble de ses données
        /// </summary>
        /// <returns>bool</returns>
        public bool AjoutBIEN(string adr, string v, int surf, int nbP, int nbC, int nbSE,
int prix, int typeB)
        {
            try
            {
                bool ok = false;
                // préparation de la requête avec des paramètres
                string requete = "insert into bien values (NULL, @adresse, @ville, @surface,
@nbP, @nbC, @nbSE, @p, @tb)";
                MySqlCommand command = myConnection.CreateCommand();
                command.CommandText = requete;

// mise à jour des paramètres de la requête préparée avec les infos passées en paramètre de
la méthode
                command.Parameters.AddWithValue("adresse", adr);
                command.Parameters.AddWithValue("ville", v);
                command.Parameters.AddWithValue("surface", surf);
```

```
                command.Parameters.AddWithValue("nbP", nbP);
                // A COMPLETER avec les paramètres manquants


                // Exécution de la requête
                int i = command.ExecuteNonQuery();
                // i est positif si l'insertion a pu avoir lieu
                return (i > 0);
            }
            catch
            {
                return false;
            }
        }
```

- Compléter cette méthode pour mettre en relation tous les paramètres de la requête préparée avec les paramètres de la méthode

- Désormais, il suffit de **compléter l'évènement Click du bouton AJOUTER** sur la vue après l'ajout dans la collection (cet évènement existe déjà, vous ajoutez à la collection acutellement, à vous de trouver le bon endroit pour compléter avec ce code pour ajouter aussi dans la BD) :

```
        // AJOUT DANS LA BD
        // appel de la méthode AjoutBD avec en paramètre les données saisies sur la vue
        bool rep = Controleur.Vmodele.AjoutBIEN(tbAdresse.Text, tbVille.Text,
Convert.ToInt32(tbSurface.Text), // paramètres à compléter dans l'ordre d'appel de la méthode….);

        if (rep)
        {
            MessageBox.Show("BIEN inséré dans la BD");
            // mise à jour des données dans le comboBox
            BtnAnnuler_Click(sender, e); // Annulation pour vider les zones de saisies
            chargerCombo(); // pour mettre à jour la comboBox
        }
        else
        {
            MessageBox.Show("Pb dans l'insertion d'un bien");
        }
```

- **Tester l'ajout d'un bien, vérifier dans la BD que le tuple s'est bien ajouté.**

BONUS : GESTION DE LA VENTE

- Modifier la base de données pour prendre en compte la gestion de la vente (et le prix de vente réel) que vous avez mise en place lors du TP4.
- Puis modifier la récupération de ces données dans le dataTable.
- Puis modifier l'insertion de cette vente dans la BD. Si le bien existe déjà, cela ne sera pas avec un INSERT mais lors d'un UPDATE que nous verrons au TP suivant TP6 (mais vous pouvez chercher comment faire).
- Il serait bien que la date de la vente soit enregistrée également, prévoir cet ajout.
- Prévoir une liste des biens vendus dans le contrôle graphique de votre choix (par une liste déroulante ou une TextBox multiplignes ou un dataGridView….).