TP7 : Programmation et BD
C# Travail sur une BD en mode déconnectée

Activités du Référentiel :

A4.1.1 Proposition d'une solution applicative
A4.1.2 Conception ou adaptation de l'interface utilisateur d'une solution applicative
A4.1.3 Conception ou adaptation d'une base de données
A4.1.7 Développement, utilisation ou adaptation de composants logiciels
A5.2.4 Étude d'une technologie, d'un composant, d'un outil ou d'une méthode

Objectifs techniques:

- ✓ Révision de la programmation selon le modèle MVC sous VisualStudio C#
- ✓ Accès aux données d'une BD (connexion – déconnexion)
- ✓ Utilisation d'un DataSet pour BD en mode déconnectée

Documents associés :

- ✓ Script de la BD : « bd_ex_bddeconnectee.sql »
- ✓ Sans oublier tout ce qui a été vu en 1ère année en SLAM2

Contexte :

- Une personne (id, nom, prenom) possède une formation de base (id, libellé).

L'application C# devra récupérer les données de la BD, puis en mode déconnecté, lister toutes les personnes, par formation et permettre l'ajout, la modification ou la suppression de personnes (CRUD sur la BD).

La base de données de ce TP sera une base de données MYSQL. Exécutez le script SQL « bd_ex_bddeconnectee.sql » fourni avec le TP.

PARTIE 1 : Environnement de développement

Pour rappel,
Dans ADO.NET, vous utilisez un objet **Connection** pour vous connecter à une source de données spécifique en fournissant les informations d'authentification nécessaires dans une chaîne de connexion. L'objet **Connection** que vous utilisez dépend du type de la source de données.

MySQL n'est pas supporté nativement dans l'environnement .NET. Mais, pour remédier à ce problème, vous pouvez télécharger différents providers qui vous permettront d'intégrer MySQL dans votre IDE.
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).

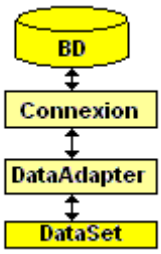
Il faudra bien ajouter les bibliothèques suivantes dès que vous utiliserez le lien avec la BD :

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

Fonctionnement d'ADO.NET : bien lire pour bien comprendre le mode déconnecté

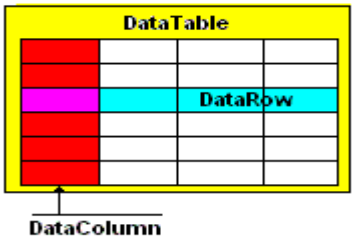
Le **DataSet est une représentation en mémoire des données.** On charge le DataSet à partir de la base de données. Une fois chargé on peut travailler en mode déconnecté. Pour effectuer une modification, on modifie le DataSet puis on met à jour la base de donnée à partir du DataSet.

Pour remplir un DataSet il faut une **Connexion** puis un **DataAdapter**.



A partir d'un dataset :

- On peut créer des DataTable 'autonomes' et y mettre une table provenant d'un DataSet:



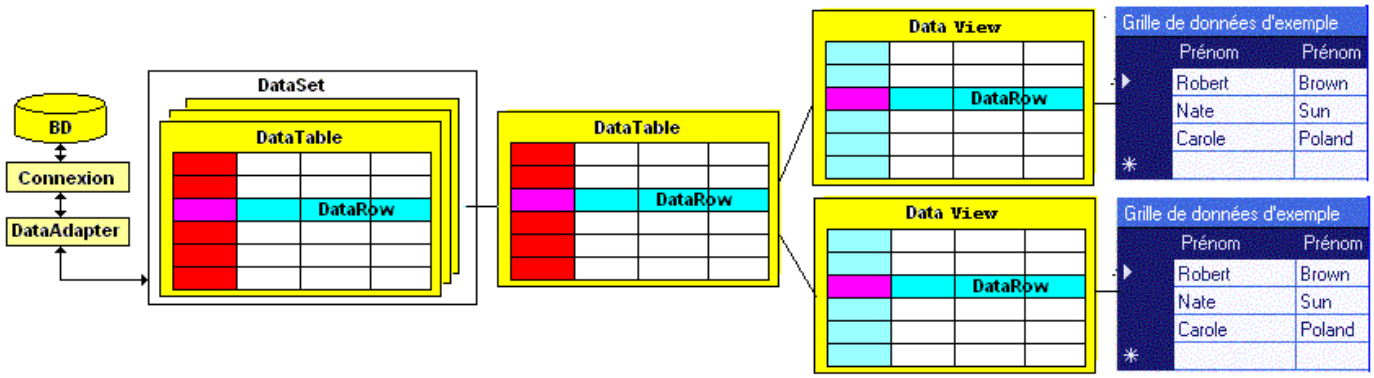
- On peut créer des DataView : Vue, représentation d'une table. A partir d'une table, on peut créer plusieurs DataView avec des représentations différentes: DataView trié, DataView dont les lignes ont été filtrées (RowFilter)

Les objets DataView sont une couche entre les objets DataTable et l'interface utilisateur. Elles permettent de présenter la table sous-jacente comme on veut. Un DataView est lié dynamiquement à la table, ce qui veut dire qu'il reflète les données et les changements apportés à la table.

Il ne faut pas confondre un DataView et une vue au sens SQL du terme. Un DataView ne peut exposer que des données provenant d'une seule table, ne peut pas créer de colonne de calcul et ne peut pas restreindre les colonnes affichées.

Enfin une DataTable ou un DataView peut être affichés dans un contrôle (datagridview par exemple).

Schéma final :



PARTIE 2 : Création de l'interface Windows Form de l'application

- Créez un nouveau projet Visual C# application Windows.

- Ajouter : Une form form1 de type MDI avec un menu :
- "Lien BD" avec 2 sous menus "import" et "export"
- « Gestion des données » avec 2 sous menus "gestion des personnes", "gestion des formations"

- Interdire l'accès à la gestion des données tant que le chargement des données utiles n'est pas réalisé (tant que l'importation dans « Lien BD » non fait).

- Ajouter : Une form « FormPersonne » contenant une datagridview (**sans maj., sans ajout, sans delete**) : cette form est appeler sur le sous-menu « gestion des personnes »

```
Multiselect: False
SelectionMode: FullRowSelect
AutoSizeColumnsMode :AllCells
AutoSizeRowsMode :AllCells
```

PARTIE 3 : Création de la classe MODELE

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

Le *Modèle* représente le comportement de l'application : traitements des données, 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.

- Ajoutez :

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

Ajoutez une variable privée private MySqlConnection myConnection;
Ajoutez une variable privée private bool connopen = false;
Ajoutez une variable privée private bool errgrave=false;
Ajoutez une variable privée private bool chargement=false;
```

- Ajoutez les accesseurs lecture de connopen, errgrave, chargement.
- Ajoutez les 2 méthodes qui permettent de se connecter et de se déconnecter (méthodes vues en 1ère année en SLAM2) :

```
public void seconnecter()
{
    string myConnectionString = "Database=bd_ex_bddeconnectee;Data Source=localhost;User Id=root;";
    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; errgrave = true;
    }
}
```

```
        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);
                errgrave = true;
            }
        }
```

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

PARTIE 4 : Création 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
- Déclarer une méthode public static init() qui permet d' instancier vmodele.
- Ajouter l'assesseur de vmodele

- Une déconnnexion si oubli :

La connexion se fera au moment de l'importation des données dans un dataSet via le modèle mais au cas où vous aurions oublié de vous déconnecter :
Gérez l'événement FormClosed pour form1 (la form MDI 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();
        }
```

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

PARTIE 5 : Connexion à la BD et importation des données

Comme étudié en partie 1, la connexion à la base de données va se faire via un DataAdapter (pour MySQL) puis remplissage d'un dataSet et d'autant de DataView que l'on aura besoin.

Nous allons donc ici déclarer **2 DataView** pour nos 2 tables (un pour la table Formation, l'autre pour la table Personne).

- Déclarer dans modele :

```
        private MySqlDataAdapter mySqlDataAdapterTP7 = new MySqlDataAdapter();
        private DataSet dataSetTP7 = new DataSet();
        private DataView dv_formation = new DataView(), dv_personne = new DataView();
```

- Créez les accesseurs pour les DataView.

- Déclarer dans modele la méthode `import ()` suivante qui va remplir le dataSet puis les dataView.
  BIEN LA COMPRENDRE POUR POUVOIR LA REUTILISER

```
        public void import()
        {
            if (!connopen) return;
            mySqlDataAdapterTP7.SelectCommand = new MySqlCommand("select * from formation;select * from
personne;", myConnection);
            try
            {
                dataSetTP7.Clear();
                mySqlDataAdapterTP7.Fill(dataSetTP7);
                MySqlCommand vcommand = myConnection.CreateCommand();

                // gestion des clés auto_increment : ici exemple sur la table personne : dataSetTP7.Tables[1]
                vcommand.CommandText = "SELECT AUTO_INCREMENT as last_id FROM INFORMATION_SCHEMA.TABLES WHERE
table_name = 'personne'";
                UInt64 der_personne = (UInt64)vcommand.ExecuteScalar();
                dataSetTP7.Tables[1].Columns[0].AutoIncrement = true;
                dataSetTP7.Tables[1].Columns[0].AutoIncrementSeed = Convert.ToInt64(der_personne);
                dataSetTP7.Tables[1].Columns[0].AutoIncrementStep = 1;

                // remplissage des dataView à partir des tables du dataSet
                dv_formation = dataSetTP7.Tables[0].DefaultView;
                dv_personne = dataSetTP7.Tables[1].DefaultView;

                chargement = true;
            }
            catch (Exception err)
            {
                MessageBox.Show("Erreur chargement dataset : " + err, "PBS formation/personne",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                errgrave = true;
            }
        }
```

- **Appel de cette méthode d'import** sur le sous-menu prévu « import ».

- 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.
- Appeler la méthode seconnecter de la classe modele via la propriété Vmodele du controleur.
- Si la connexion n'a pas eu lieu (propriété Connopen de modele à false) : indiquer un messageBox
- Sinon (connexion ok) :
  - appeler la méthode import de la classe modele via la propriété Vmodele du controleur.
  - Si le chargement est ok (propriété Chargement à true) : rendre accessible le menu « gestion des données accessibles »
- Appeler la méthode sedeconnecter de la classe modele via la propriété Vmodele du controleur.

- **Tester et vérifier que la connexion se passe bien et le chargement des dataView aussi.**

La BD est désormais chargée dans des DataSet et DataView. Nous les allons les utiliser sans atteindre directement la BD qui se trouve déconnectée.

PARTIE 6 : Gestion des données : remplissage des données dans un dataGridView

Sur la form « gestion des personnes », vous avez mis un contrôle de type DataGridView.
Nous allons charger les personnes dans ce dataGridView en utilisant le dataView créé en partie précédente.

- Déclarer en privé de cette form :

```
private BindingSource bindingSource1 = new BindingSource();
```

- Ajouter une méthode chargedgv() qui va remplir le dataGridView (appelé ici dataGV) à partir du dataView des personnes de la classe modele (appelé ici Dv_personne)

```
public void chargedgv()
{
    bindingSource1.DataSource = controleur.Vmodele.Dv_personne;
    dataGV.DataSource = bindingSource1;
}
```

- Ajouter au constructeur de cette form l'appel à cette méthode `chargedgv()`.
- Ne pas oublier de faire l'ouverture de cette form sur le sous-menu…
- **Tester avec votre serveur de base de données (wamp) démarré tout au long de l'éxécution**
- **Puis tester en stoppant votre serveur de BD après l'import**. Que constatez-vous ?

- Amélioration de l'affichage dans le dataGridView : rendre des colonnes invisibles, gérer la taille des colonnes, gérer les barres de défilement :

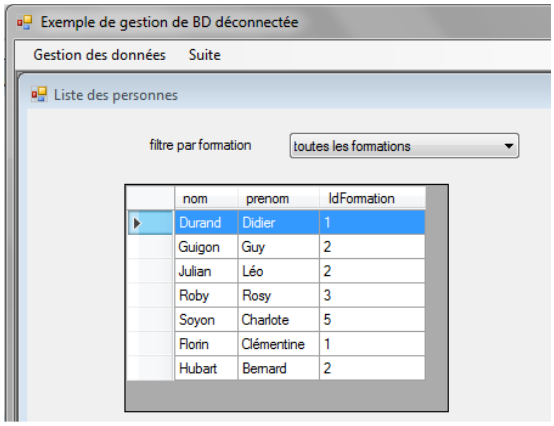
Par exemple, à ajouter dans la méthode `chargedgv()` :

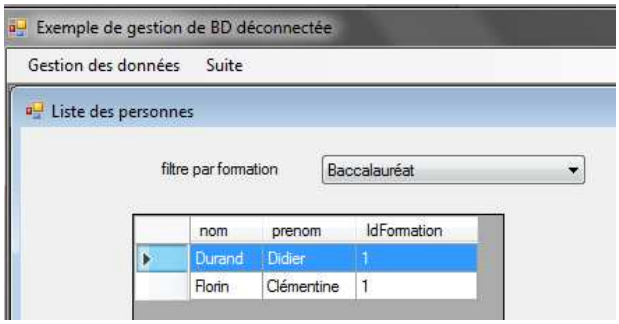
```
dataGV.Columns[0].Visible = false; // pour rendre invisible la colonne des IdPersonne

int vwidth = dataGV.RowHeadersWidth;
for (int i = 0; i < dataGV.Columns.Count; i++)   // pour gérer la taille des colonnes
{
    if (dataGV.Columns[i].Visible)
        vwidth = vwidth +
          dataGV.Columns[i].GetPreferredWidth(DataGridViewAutoSizeColumnMode.AllCells, false);
}
dataGV.Width = vwidth;
if (dataGV.ScrollBars.Equals(ScrollBars.Both) | dataGV.ScrollBars.Equals(ScrollBars.Vertical))
{
    dataGV.Width += 20;
}
dataGV.Refresh();
```

➢ **<u>Ajout de filtre</u>** : affichage des personnes selon la formation

Toutes les personnes sont affichées ici dans ce dataGridView. Nous allons ajouter une liste déroulante avec les formations qui permettra de sélectionner une formation et de n'afficher que les personnes correspondantes à cette formation.





- Ajouter à la form une combobox cbFormations
- Charger cette combobox dans la méthode chargedgv() via :

```
cbFormations.Items.Clear();

// creation d'une liste des formations
List<KeyValuePair<int, string>> FList = new List<KeyValuePair<int, string>>();
FList.Add(new KeyValuePair<int, string>(0, "toutes les formations"));
cbFormations.Items.Add("toutes les formations");

// on parcourt le dataView des formations Dv_formation de la classe modele pour compléter la FList
for (int i = 0; i < controleur.Vmodele.Dv_formation.ToTable().Rows.Count; i++)
{
   FList.Add(new KeyValuePair<int,
         string>((int)controleur.Vmodele.Dv_formation.ToTable().Rows[i][0],
         controleur.Vmodele.Dv_formation.ToTable().Rows[i][1].ToString()));
}
// on lie la liste à la comboBox
cbFormations.DataSource = FList;
cbFormations.ValueMember = "Key";
cbFormations.DisplayMember = "Value";
cbFormations.Text = cbFormations.Items[0].ToString();

cbFormations.DropDownStyle = ComboBoxStyle.DropDownList;
```

- **Tester** et vérifier que les formations se chargent bien dans la comboBox
- Ajouter une méthode changefiltre() à la form :

```
private void changefiltre()
{
    string num = cbFormations.SelectedValue.ToString();   // on récupère la formation sélectionnée

    int n =  Convert.ToInt32(num);
    if (n == 0) // cas de "toutes les formations"
        controleur.Vmodele.Dv_personne.RowFilter = "";  // pour annuler le filtre
    else
    {
        string Filter = "IdFormation = '" + n + "'";     // filter sur IdFormation
        controleur.Vmodele.Dv_personne.RowFilter = Filter;   // on applique le filter à la dataView
    }
    dataGV.Refresh();   // mise à jour du dataGridView
}
```

- Sur l'événement SelectionChangeCommitted de la comboBox faire un appel à changefiltre().

<u>Aide pour faire des filtres avec multi-conditions</u> :

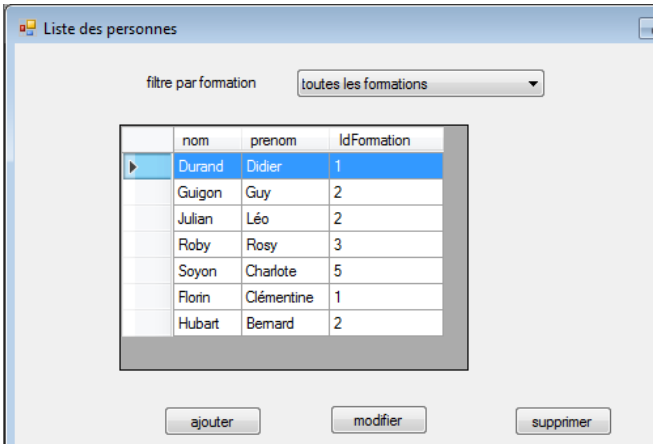
```
string Filter = "nomcolonne<= '" + nomvariable + "' AND (nomcolonne is Null OR nomcolonne > '" +
nomvariable + "')";
```

➢ **Tester en mettant un filtre ou en le retirant.**

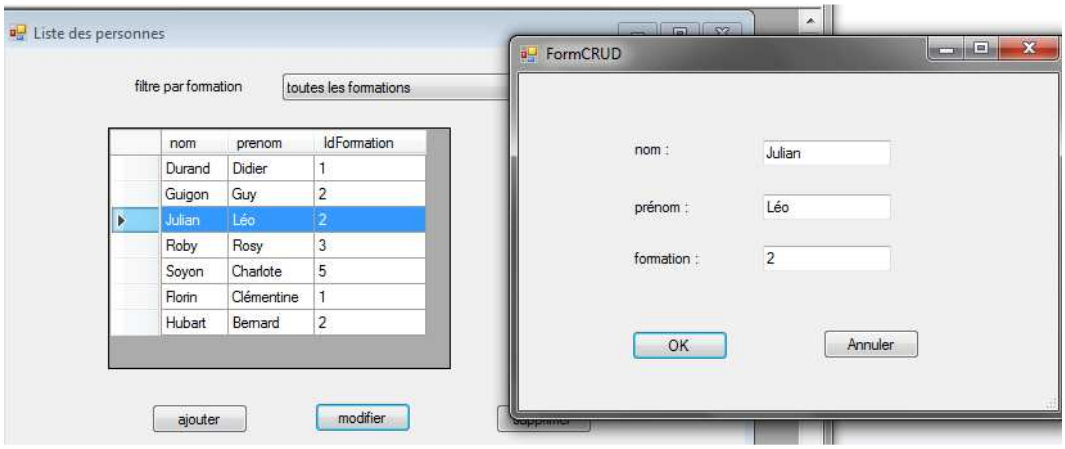
PARTIE 7 : Modification des données dans les dataView : ajout/modification/suppression

➢ Mise en forme de la vue :

- Ajouter 3 boutons pour ajouter/modifier/supprimer sur la form « Gestion des personnes »



A la sélection de l'un des 3 boutons, on affichera une nouvelle forme qui permettra, soit l'ajout d'une personne (champs vides au départ), soit la modification ou la suppression de la personne sélectionnée dans le dataGridView (champs remplis avec les données dès l'ouverture).



➢ Créer cette nouvelle form « `formCRUD` » avec 3 zones de texte et 2 boutons.

Cette form sera de type dialogBox donc

- Modifier la propriété `DialogResult` du bouton OK en mettant : OK
- Modifier la propriété `DialogResult` du bouton Annuler en mettant : Cancel
- Modifier la propriété `AcceptButton` de la form en sélectionnant votre bouton OK
- Modifier la propriété `CancelButton` de la form en sélectionnant votre bouton Annuler

➢ Cette form sera gérée à partir de la précédente donc pour pouvoir accéder aux zones de texte (pour pouvoir les remplir si modif ou suppression), il faut gérer leurs accesseurs dans le fichier `formCRUD.designer.cs`. Générer les accesseurs des 3 zones de texte. Bien sauvegarder.

➢ Créer une méthode `crud_personne` dans le contrôleur qui va gérer la **mise à jour du dataView.** Il faut bien se rappeler que nous sommes en mode déconnecté donc la BD ne sera pas à jour directement (on verra cela dans la partie suivante avec l'export).

Cette méthode sera la même que l'on soit en ajout/modif ou suppression, donc on précisera en paramètre l'action à faire sur le dataView.

```
public static void crud_personne(Char c, String cle)
{
// Le Char c correspond à l'action : c:create, u update, d delete,
// la cle est celle de l'enregistrement sélectionné, vide si action d'ajout (c = 'c')


    int index = 0;
    FormCRUD formCRUD = new FormCRUD();  // création de la nouvelle forme
    if (c== 'c')  // mode ajout donc pas de valeur à passer à la nouvelle forme
    {
        // à écrire : mettre les zones de texte de formCRUD à vide
    }

    if (c=='u' || c=='d')   // mode update ou delete donc on récupère les champs
    {

        string sortExpression = "IdPersonne";
        vmodele.Dv_personne.Sort = sortExpression;   // on trie le DataView sur les IdPersonne

        // on recherche l'indice où se trouve la personne sélectionnée
        // grâce à la valeur passée en paramètre donc grâce à son Id
        index = vmodele.Dv_personne.Find(cle);

        // on remplit les zones par les valeurs du dataView correspondantes
        formCRUD.TbNom.Text = vmodele.Dv_personne[index][1].ToString();
        formCRUD.TbPrenom.Text = ; // à compléter
        formCRUD.TbFormation.Text = ; // à compléter
    }
        // on affiche la nouvelle form
        formCRUD.ShowDialog();

        // si l'utilisateur clique sur OK
        if (formCRUD.DialogResult == DialogResult.OK)
        {
            if (c=='c') // ajout
            {
                // on crée une nouvelle ligne dans le dataView
                DataRowView newRow = vmodele.Dv_personne.AddNew();
                newRow["IdPersonne"] = 15;
                newRow["nom"] = formCRUD.TbNom.Text;
                newRow["prenom"] = ; // à compléter
                newRow["IdFormation"] = ; // à compléter
                newRow.EndEdit();
            }

            if (c=='u')  // modif
            {
                // on met à jour le dataView avec les nouvelles valeurs
                vmodele.Dv_personne[index]["nom"] = formCRUD.TbNom.Text;
                // à compléter pour la mise à jour des autres champs
            }

            if (c == 'd')  // suppression
            {
                // on supprime l'élément du DataView
                vmodele.Dv_personne.Table.Rows[index].Delete();
            }

            MessageBox.Show("OK : données enregistrées");
            formCRUD.Dispose();  // on ferme la form
        }
        else
        {
            MessageBox.Show("Annulation : aucune donnée enregistrée");
            formCRUD.Dispose();
        }
}
```

- Il ne reste plus qu'à appeler cette méthode `crud_personne` avec les bons paramètres sur les évènements click des boutons AJOUTER/MODIFIER/SUPPRIMER.

Exemple pour le bouton MODIFIER : Vous en inspirer pour AJOUTER et SUPPRIMER.

```
private void btnModifier_Click(object sender, EventArgs e)
{
    // vérifier qu'une ligne est bien sélectionnée dans le dataGridView
    if (dataGV.SelectedRows.Count == 1)
```

```
        {
            // appel de la méthode du controleur en mode update et avec la valeur de l'IdPersonne en clé
            controleur.crud_personne('u', dataGV.Rows[dataGV.SelectedRows[0].Index].Cells[0].Value.ToString());

            // mise à jour du dataGridView en affichage
            bindingSource1.MoveLast();
            bindingSource1.MoveFirst();
            dataGV.Refresh();
        }
        else
        {
            MessageBox.Show("Sélectionner une ligne à modifier");
        }
    }
```

- **Tester les 3 actions.**

PARTIE 8 : L'export : modification dans la base de données !

Un peu de théorie : Les accès concurrents et l'intégrité des données

En mode déconnecté toutes les modifications n'ont pas été répercutées dans la BDD. Le Dataset n'est qu'un curseur statique côté client et surtout il est déconnecté.
Lorsqu'un Dataset est modifié et qu'on lui demande de répercuter ses modifications vers la source de données, celui-ci gère la concurrence par un verrouillage optimiste, qui consiste à une comparaison de valeurs. Il existe grosso modo trois gestions possibles.

- **Récupération d'une valeur de version** : Ceci consiste à utiliser une donnée qui est intrinsèque à la ligne mais qui n'est pas une donnée de l'enregistrement. Par exemple une valeur TimeStamp de la dernière modification. Lors de l'appel de la mise à jour, il y a comparaison de la valeur de l'enregistrement et de celle stockée dans le Dataset. Si elles sont différentes, c'est à dire si un autre utilisateur a modifié l'enregistrement, il y a conflit de la concurrence optimiste, une exception est générée et la mise à jour est refusée.

- **Comparaison des enregistrements** : C'est le principe utilisé par le moteur de curseur client. Elle consiste à comparer les valeurs des champs trouvées dans le Dataset à celles présentes dans la source. Elle ne porte pas forcément sur l'ensemble des valeurs. Le principe réel est simple on exécute une requête action contenant dans la clause WHERE les valeurs de champs présentes dans le Dataset, et on récupère le nombre d'enregistrements modifiés. Si celui-ci est nul, c'est que l'enregistrement a été modifié et qu'il y a concurrence optimiste. Cette technique peut être périlleuse si on n'utilise pas tous les champs dans la requête, mais c'est une question de choix.

- **Méthode de barbare que les anglophones nomment 'last in win'** qui consiste à ne pas gérer la concurrence et à toujours écraser l'enregistrement par les dernières modifications arrivant. Attention aux effets de ce genre de stratégie (notamment si plusieurs utilisateurs en connexion simultanée) !

Notions de transactions :

Une **transaction** transforme une base de données (ou tout autre objet) d'un état consistant à un autre état consistant. Les transactions se déroulent de façons isolées, protégées des autres transactions concurrentes.

Une **transaction** assure que la *totalité* de la tâche soit effectuée de façon définitive (*commit*), sinon *rien* ne soit fait (*rollback*, en cas d'erreur par exemple).
*Par exemple, lors d'un virement bancaire, un compte est d'abord débité, puis un autre est crédité. Imaginons, qu'après avoir débité le premier compte et avant que le deuxième soit crédité, un incident se produit, arrêtant le processus. Dans ce cas, la **transaction** doit assurer que le premier compte reprenne son état initial.*

La stratégie que l'on applique pour la concurrence doit être définie en toute connaissance de cause. En choisir une au hasard revient à ne pas gérer la concurrence.

Cas particulier : Delete or not delete

Imaginons le cas d'un enregistrement dont un utilisateur demande une modification mais entre-temps, un autre utilisateur en demande la suppression : que faire ? faut-il modifier un enregistrement qui a été déjà supprimé ?

Devons-nous arrêter une transaction de mise à jour (update ou delete) si la tentative de modification ou de delete d'un enregistrement se heurte à un enregistrement déjà supprimé par un autre utilisateur?

Conseil :

- Pour un delete la transaction ne s'arrête pas
- Pour un update la transaction s'arrête, toutes les MAJ de cet enregistrement sont annulées le DataSet est rechargé et l'utilisateur en ait informé

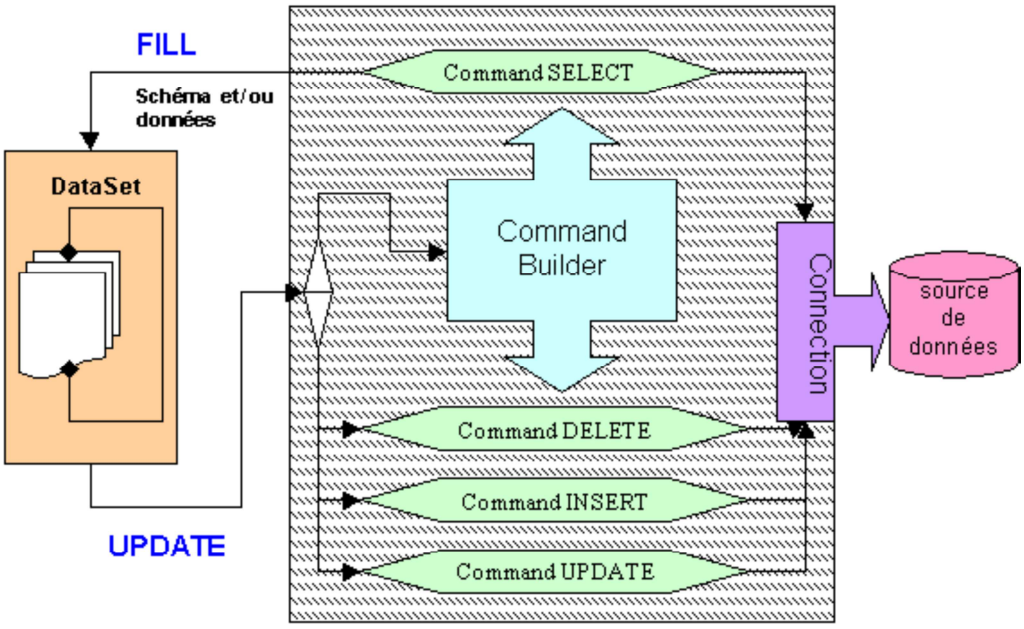
En tant que développeur, vous devez donc **prendre une décision** concernant :

- L'update d'un enregistrement déjà maj
- L'update d'un enregistrement supprimé

Dans la pratique en C# : méthode « comparaison des enregistrements »

Comme nous l'avons déjà dit, il n'y a jamais de connexion directe entre le DataSet et la source de données. Le DataSet ne peut communiquer qu'avec le DataAdapter à l'aide des méthodes que celui-ci fournit.

Le DataAdapter quant à lui communique avec la source en utilisant un objet Connection et quatre objets Command (DeleteCommand, UpdateCommand, InsertCommand).



Le DataAdapter ne génère pas spontanément le code de mise à jour. Il attend que vous lui fournissiez les objets Command nécessaire à la mise à jour dans ses objets UpdateCommand, DeleteCommand et InsertCommand.

L'exportation des données du DataSet local vers la BD va se faire sur le sous-menu « Export » du menu « Lien BD » de la form1 principale.

- Dans la classe « modele », ajouter :

```
using System.Collections;
private bool errmaj = false;
private char vaction, vtable;
private ArrayList rapport = new ArrayList(); // pour gérer le rapport des erreurs de maj
```

- Générez les accesseurs de ces propriétés.

- Ajouter la **méthode de gestion des erreurs** de mise à jour (à aouter dans modele) :

Cette méthode va se déclencher au moment de chaque évènement de demande de modification sur une table.
Vaction (de la classe modele) précise la modification en cours (c : create, u : update, d : delete)
Vtable (de la classe modele) précise la table concernée (ici une seule table donc vtable == 'p' pour la table PERSONNE).

BIEN LA COMPRENDRE POUR POUVOIR LA REUTILISER et la compléter (si modifications sur plusieurs tables par exemple)

```
        private void OnRowUpdated(object sender, MySqlRowUpdatedEventArgs args)
       // utile pour ajout,modif,supp
        {
            string msg="";
            Int64 nb = 0;
            if (args.Status == UpdateStatus.ErrorsOccurred)
            {
                if (vaction == 'd' || vaction == 'u')
                {
                    MySqlCommand vcommand = myConnection.CreateCommand();
                    if (vtable == 'p')  // 'p' pour table PERSONNE
                    {
                        vcommand.CommandText = "SELECT COUNT(*) FROM personne WHERE IdPersonne = '" +
args.Row[0, DataRowVersion.Original] + "'";
                    }
                    nb = (Int64)vcommand.ExecuteScalar();
                    // on veut savoir si la personne existe encore dans la base
                }
                if (vaction == 'd')
                {
                    if (nb == 1) // pour delete si l'enr a été supprimé on n'affiche pas l'erreur
                    {
                        if (vtable == 'p')
                        {
                            msg = "pour le numéro de personnes : " + args.Row[0, DataRowVersion.Original] + "
impossible delete car enr modifié dans la base";
                        }
                        rapport.Add(msg);
                        errmaj = true;
                    }
                }
                if (vaction == 'u')
                {
                    if (nb == 1)
                    {
                        if (vtable == 'p')
                        {
                            msg = "pour le numéro de personne: " + args.Row[0, DataRowVersion.Original] + "
impossible MAJ car enr modifié dans la base";
                        }
                        rapport.Add(msg);
                        errmaj = true;
                    }
                    else
                    {
                        if (vtable == 'p')
                        {
                            msg = "pour le numéro de personne : " + args.Row[0, DataRowVersion.Original] + "
impossible MAJ car enr supprimé dans la base";
                        }
                        rapport.Add(msg);
                        errmaj = true;
                    }
                }
                if (vaction == 'c')
                {
                    if (vtable == 'p')
                    {
                        msg = "pour le numéro de personne : " + args.Row[0, DataRowVersion.Current] + "
impossible ADD car erreur données";
                    }
                    rapport.Add(msg);
                    errmaj = true;
                }
            }
        }
```

- Puis ajouter (toujours dans modele) une **méthode par action de mise à jour**. Nous aurons donc une méthode pour l'ajout, une méthode pour la modification, une méthode pour la suppression.

  Chaque méthode :

- Précise l'action (vaction) et la table concernée (vtable)
- Lance un appel à la méthode précédente pour la gestion des éventuels conflits
- Crée la commande du DataAdapter (insert, update ou delete selon l'action). Cette commande se fera par une requête préparée avec les champs de la table
- Déclare les paramètres utiles au déclenchement de la commande
- Précise la table concernée par le DataSet
- Ajoute les enregistrements reconnus en local
- Supprime l'appel à la méthode de gestion des conflits

Par exemple, pour **l'action d'ajout** :

```
public void add_personne()
{
    vaction = 'c';  // on précise bien l'action, ici c pour create
     vtable = 'p';
    if (!connopen) return;
    //appel d'une méthode sur l'événement ajout d'un enr de la table
    mySqlDataAdapterPPE4.RowUpdated += new MySqlRowUpdatedEventHandler(OnRowUpdated);
    mySqlDataAdapterPPE4.InsertCommand = new MySqlCommand("insert into personne (nom,prenom,IdFormation) values
    (?nom,?prenom,?IdFormation)", myConnection);// notre commandbuilder ici ajout non fait si erreur de données

    //déclaration des variables utiles au commandbuilder
    // pas besoin de créer l'IdPersonne car en auto-increment
    mySqlDataAdapterPPE4.InsertCommand.Parameters.Add("?nom", MySqlDbType.Text, 65535, "nom");
    mySqlDataAdapterPPE4.InsertCommand.Parameters.Add("?prenom", MySqlDbType.Text, 65535, "prenom");
    mySqlDataAdapterPPE4.InsertCommand.Parameters.Add("?IdFormation", MySqlDbType.Int16, 10, "IdFormation");

    //on continue même si erreur de MAJ
    mySqlDataAdapterPPE4.ContinueUpdateOnError = true;

    //table concernée 1 = personne
    DataTable table = dataSetPPE4.Tables[1];

    //on ne s'occupe que des enregistrement ajoutés en local
    mySqlDataAdapterPPE4.Update(table.Select(null, null, DataViewRowState.Added));

    //désassocie la méthode sur l'événement maj de la base
    mySqlDataAdapterPPE4.RowUpdated -= new MySqlRowUpdatedEventHandler(OnRowUpdated);
}
```

- Vous inspirez de cette méthode `add_personne()` pour créer la **méthode `maj_personne()`** et **`del_personne()`.**

Aide :

- Pour la mise à jour, la commande du commandBuilder devient :

```
            mySqlDataAdapterPPE4.UpdateCommand = new MySqlCommand("update personne set
nom=?nom,prenom=?prenom, IdFormation=?IdFormation where IdPersonne = ?num  ", myConnection);
```

- Pour la suppression, la commande du commandBuilder devient :

```
            mySqlDataAdapterPPE4.DeleteCommand = new MySqlCommand("delete from personne where IdPersonne =
?num;", myConnection);// force le delete même si maj dans la base
```

- Reste plus qu'à créer la **méthode `export()`** dans modele qui appelle ces 3 méthodes. Bien vérifier auparavant que la connexion à la BD est faite (`if (!connopen) return;`)

Un `try catch` peut être utile pour gérer les éventuelles erreurs d'exécution.

- Appeler cette méthode d'exportation sur le sous-menu « Export » du menu « Lien BD » de la form1 principale :

```
private void exportToolStripMenuItem_Click(object sender, EventArgs e)
        {
              // se connecter à la BD par le biais de Vmodele du controleur
              // si la connexion échoue : propriété connopen de vmmodele à faux
                    // messageBox d'erreur
              // sinon
                    // appel de la méthode export() via Vmodele du controleur
              // se déconnecter de la BD.
        }
```

- **Tester les 3 actions avec ajout, modif et suppression en local puis avec export.**

PARTIE 9 : Et avec la gestion de la date de mise à jour du champ !

Dans la pratique en C# : méthode « Récupération d'une valeur de version »

Comme vu en explications de la partie 8, il existe plusieurs méthodes pour la mise à jour de données.
La méthode de « récupération d'une valeur de version » va permettre de comparer par rapport à une date et heure de dernière modification.

- Dans la BD, **modifier la structure de la table PERSONNE** en ajoutant un champ date de type timestamp (par défaut current, oupdate current) :

  ```
  `dateA` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
  ```

  Attention : Champ not null : donc modifier vos enregistrements précédents si besoin.

- Dans la classe modele :
- Pas de modification pour l'ajout puisque par défaut la date et heure courantes seront ajoutées
- **Pour la modification et suppression** : la commande va changer pour modifier aussi la date en ajoutant un paramètre à notre requête update ou delete :

Exemple pour update dans la méthode `maj_personne()` :

```
mySqlDataAdapterPPE4.UpdateCommand = new MySqlCommand("update personne set
nom=?nom,prenom=?prenom, IdFormation=?IdFormation where IdPersonne = ?num and dateA = ?modif" ,
myConnection);

// ajout du paramètre modif
mySqlDataAdapterPPE4.UpdateCommand.Parameters.Add("?modif", MySqlDbType.Timestamp, 20, "dateA");
```

- A adapter pour la méthode `del_personne()`.

- **Tester les 3 actions avec ajout, modif et suppression en local puis avec export.**