Hiver 2020

Introduction à ADO.NET

Saliha Yacoub
COLLÈGE LIONEL-GROULX

Table des matières

Chapitre1, introduction

ADO.NET est un ensemble de classes qui exposent des services standardisés d'accès aux données. Ces classes permettent donc aux programmeurs de concevoir des applications permettant de se connecter à des sources de données variées et, d'extraire, de manipuler et de mettre à jour ces données. Une des principales caractéristiques de l'architecture d'ADO.NET, est le fait qu'elle intègre deux modèles d'accès aux données qui sont :

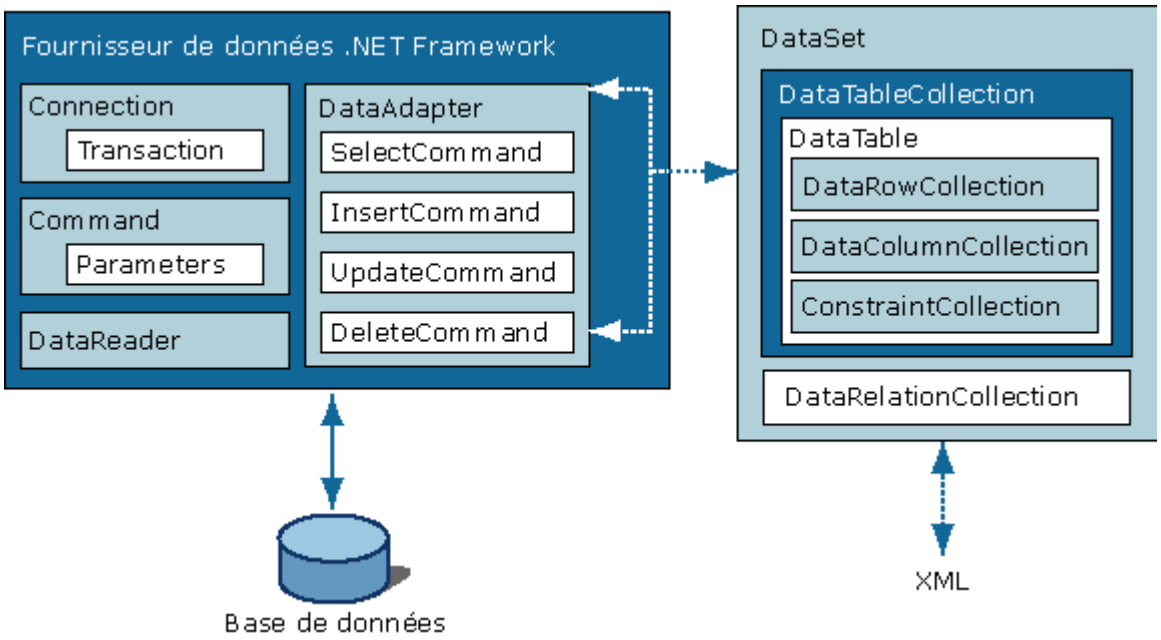
- Le modèle « **connecté** » qui est bien adapté aux applications à deux couches traditionnelles;
- Le modèle « **déconnecté** » qui est destinés aux applications multicouches en utilisant un DataSet.

Les sources de données peuvent être :

- des SGBD relationnels tels **Microsoft SQL Server** et **Oracle**
- des sources de données exposées via **OLE DB**.
- des sources de données exposées via **XML**.

Les composants de ADO.NET ont été conçus de façon à distinguer l'accès aux données de la manipulation de données. Cette distinction est rendue possible par deux composants centraux de ADO.NET : le **DataSet** et le **fournisseur de données**.

Le schéma suivant représente les composants de l'architecture ADO.NET.



Le fournisseur de données

Un fournisseur de données est utilisé pour :

- La connexion à une base de données ;
- L'exécution de commandes;
- L'extraction de résultats.

En ADO.NET, les principaux fournisseurs de données sont les suivants :

- Fournisseur de données « .NET Framework » pour SQL Server ;
- Fournisseur de données « .NET Framework » pour OLE DB;
- Fournisseur de données « .NET Framework » pour ODBC ;
- Fournisseur de données « Oracle Data Provider pour le NET (ODP.NET) » pour Oracle

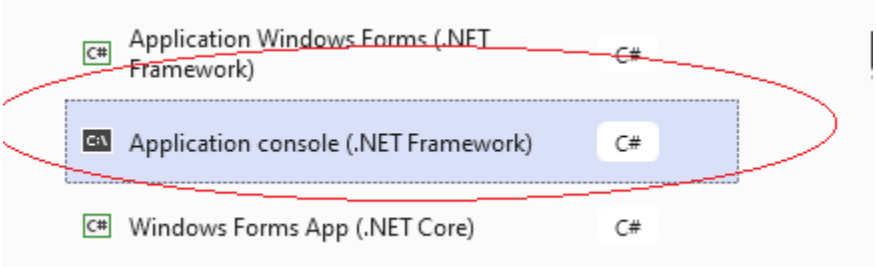
Avec ADO.NET, le fournisseur de données est conçu pour être léger et créer une couche minimale entre la source de données et votre code, afin d'augmenter les performances sans réduire la fonctionnalité. Il se comprend un ensemble de composants comprenant les objets **Connection**, **Command**, **DataReader** et **DataAdapter**.

Ces composants sont explicitement conçus pour la manipulation des données et un accès aux données rapide. L'objet **Connection** assure la connectivité avec une source de données. L'objet **Command** permet l'accès aux commandes de base de données pour retourner des données, modifier des données, exécuter des procédures stockées et envoyer ou extraire des informations sur les paramètres. Le **DataReader** fournit un flux très performant de données en provenance de la source de données. Enfin, le **DataAdapter** établit une passerelle entre l'objet **DataSet** et la source de données. Le **DataAdapter** utilise les objets **Command** pour exécuter des commandes SQL au niveau de la source de données afin d'une part d'approvisionner le **DataSet** en données, et d'autre part de répercuter dans la source de données les modifications apportées aux données contenues dans le **DataSet**.

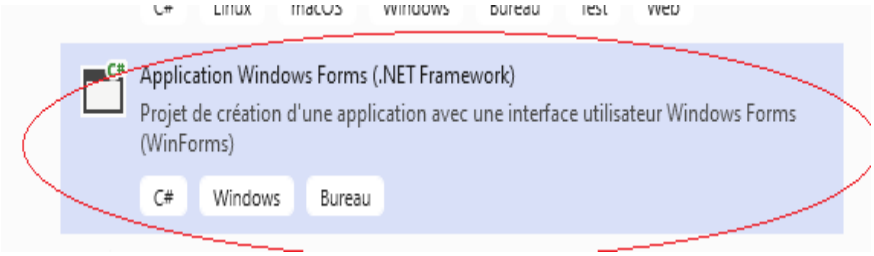
Le fournisseur de données SqlClient fait partie intégrante de .NET Framework. Pas besoin d'installation supplémentaire (comme ODAC ou ODP.NET pour Oracle) si vous voulez accéder à votre base de données SQL Server par C#, VB.net ou ASP.Net (voir figure suivante)

```
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;

namespace AdoSqlServer
{
    0 références
    class Program
    {
        0 références
        static void Main(string[] args)
        {
            try
            {
                string chaineConnexion = "Data source =M-INFO-SY\\
                SqlConnection connSql = new SqlConnection();
```

En mode console, vous avez ce type d'application C#



En mode graphique :



Chapitre2, présentation de l'objet SqlConnection

Un objet SqlConnection représente une connexion à la base de données SQL Server. Il a donc pour rôle d'établir une connexion à la base de données.
Les connexions sont utilisées pour « parler » aux bases de données et sont présentées par la classe SqlConnection.

On crée un objet SqlConnection avec la méthode **new()**

```
SqlConnection connSql = new SqlConnection();
```

Propriétés importantes de l'objet SqlConnection

Propriété ConnectionString

Il s'agit de la chaîne de connexion qui comprend des paramètres requis pour établir la connexion initiale. La valeur par défaut de cette propriété est une chaîne vide ("").
Les principaux paramètres pouvant être inclus dans la chaîne sont les suivants :

- Data source : le nom de serveur ou de la source de données;
- Initial Catalog = nom de la base de données
- User Id : Compte de connexion SQL Server ;
- Password : Mot de passe de la session du compte SQL server

Attention : 

Une chaine de connexion est toujours un string et de la forme (remarquez les points virgule;

```
string chaineConnexion = "Data source =nomDuserveur; Initial Catalog =nomDB; User
Id= unUser;password =mpasse";
```

En d'autres mots, il faudra fournir au système l'ensemble des paramètres que vous fournissez lorsque vous utilisez SSMS

Exemple: (sur une seule ligne)

```
string chaineConnexion = "Data source =M-INFO-SY\\SQLEXPRESS;
Initial Catalog =EmpClgDB; User Id= Patoche;password =remi2002";
```

Pour passer la chaine de connexion, on utilise la propriété **`ConnectionString`** de l'objet SqlConnection.

`connSql.ConnectionString = ChaineConnexion;` (connSql étant définit plus haut)

Propriété State.

Indique l'état actuel de de la connexion. Nous avons deux valeurs pour cette propriété. Closed ou Open. Par défaut la valeur est Closed

Méthodes importantes :

La méthode Open()

Permet d'ouvrir une connexion à la base de données

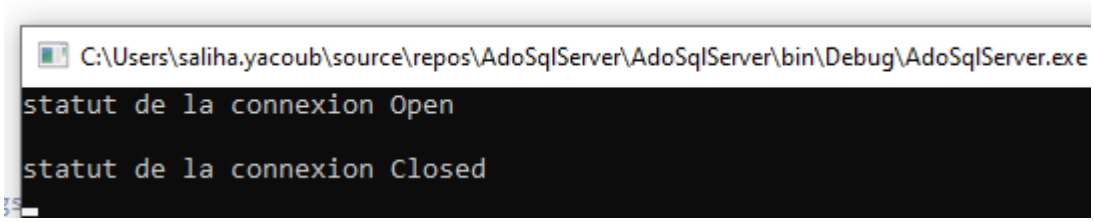
La méthode Close()

Permet de fermer une connexion ouverte à la base de données.

Exemple

```
namespace AdoSqlServer
{
    class Program
    {
        static void Main(string[] args)
        {



            SqlConnection connSql = new SqlConnection();

              try
                  {
                   string chaineConnexion = "Data source =M-INFO-SY\\SQLEXPRESS;
                   Initial Catalog =EmpClgDB; User Id= Patoche;password =remi2002";

                   connSql.ConnectionString = chaineConnexion;
                   connSql.Open();

                   Console.WriteLine("statut de la connexion" + " " + connSql.State);
                   Console.Read();

                    }

             catch (Exception ex)
                { Console.WriteLine(ex.Message);
                  Console.Read();
                 }

            connSql.Close();
            Console.WriteLine("statut de la connexion" + " " + connSql.State);
            Console.Read();
        }
    }
}
```

Ce qui donne ceci :



Chapitre 3, présentation de l'objet SqlCommand

L'objet SqlCommand contient les commandes envoyées aux SGBD. Ces commandes sont envoyées soit en utilisant des requêtes simples, soit en utilisant des procédures stockées. Lorsque la requête SQL ou procédure retourne un résultat, il est retourné dans un SqlDataReader ou autre (voir **plus loin).**

L'objet SqlCommand se crée avec la méthode **new()**

Attention : 

Pour envoyer une requêtre à la base de donnée, on utilise un objet SqlCommad, en fournissant une connexion et une requête SQL ou une procédure stockeés.

Nous présenterons rapidement les cas des requêtes simples et nous allons nous attarder un peu plus sur les appels de procédures stockées.

SqlCommand possède plusieurs constructeurs, pour commencer, celui qu'on va utiliser le constructeur suivant :

SqlCommand(string SQL , SqlConnection connSql)

Exemple : (connSql étant le nom de l'objet SqlConnection)

```
string sql1 = "update employes set salaire = 10 where EMPNO = 11";

SqlCommad sqlCmd = new SqlCommad(sql1, connSql);
```

Quelques propriétés de l'objet SqlCommad

CommandText	Obtient ou définit l'instruction SQL ou la procédure stockée à exécuter sur la base de données
CommandType	Obtient ou définit une valeur indiquant la manière dont la propriété **CommandText** doit être interprétée (instruction SQL ou **procédure)**
Connection	Obtient ou définit l'objet **SqlConnection** utilisé par cette instance de **SqlCommand**.
Parameters	Spécifie les paramètres de la requête SQL ou **de la procédure stockée**

Méthodes importantes de l'objet SqlCommad

ExecuteNonQuery()	Exécute une instruction SQL sur **`Connection`** et retourne le nombre de lignes affectées. Pour toutes les requêtes DML
ExecuteReader()	Surchargé. Envoie **`CommandText`** à **`Connection`** et génère Sql**`DataReader.`** `Pour les requêtes SELECT`
ExecuteScalar()	Exécute la requête et retourne la première colonne de la première ligne. (en général pour les SELECT COUNT, SELECT MIN …

La méthode ExecuteNonQuery()

Cette méthode permet d'exécuter une requête DML (INSERT, UPDATE et DELETE). Cette méthode retourne un **int**, indiquant le nombre de lignes affectées par la requête.

La méthode ExecuteReader()

Cette méthode permet d'exécuter une requête SELECT et retourne un **SqlDataReadr** contenant le résultat de la requête.

La méthode ExecuteScalar()

Exécute la requête et retourne la première colonne de la première ligne. Elle est utilisée pour des requêtes comme SELECT COUNT, SELECT MIN etc…

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;

namespace AdoSqlServer
{
    class Program
    {
        static void Main(string[] args)
        {


            SqlConnection connSql = new SqlConnection();


            try
                  {
                   string chaineConnexion = "Data source =M-INFO-SY\\SQLEXPRESS; Initial Catalog
                    =EmpClgDB; User Id= Patoche;password =remi2002";
                   connSql.ConnectionString = chaineConnexion;
                   connSql.Open();
                   Console.WriteLine("statut de la connexion" + " " + connSql.State);
                   Console.Read();
              //Jusque là nous avons établie une connexion.
              // On passe une requête SQL, INSERT
          string sql = "insert into employesClg(nom, prenom, typeEmploye) values('ADO', 'NET','P')";
                SqlCommand sqlCommd = new SqlCommand(sql, connSql);
                int nb_lignes = sqlCommd.ExecuteNonQuery();
                Console.WriteLine("nombre de ligne" + nb_lignes);
                Console.Read();
            }

            catch (Exception ex)
                { Console.WriteLine(ex.Message);
                Console.Read();
                 }
             connSql.Close();
             Console.WriteLine("statut de la connexion" + " " + connSql.State);
             Console.Read();
        }
    }

}
```

Chapitre 4, présentation de l'objet SqlDataReader

Les objets **DataReader** servent à extraire d'une base de données un flux de données en lecture seule et dont le défilement se fera par en avant uniquement (read-only, forward-only,). Les résultats sont retournés pendant que la requête s'exécute et stockés dans la mémoire tampon de réseau sur le client jusqu'à ce que vous les demandiez au moyen de la méthode Read de **DataReader**.

L'objet SqlDataReader se crée par la méthode **ExecuteReader()** de l'objet **SqlCommad**

Attention : 

L'objet OraceDataReader NE SE Crée pas ave la méthode new().

Quelques propriétés importantes de SqlDataReader

FieldCount	Obtient le nombre de colonnes figurant dans la ligne en cours.
HasRows	Obtient une valeur indiquant si **`SqlDataReader`** contient une ou plusieurs lignes.
IsClosed	Indique si **`SqlDataReader`** est fermé.

Quelques méthodes importantes de SqlDataReader

Colse()	Ferme l'objet SqlDataReader
Dispose ()	Libère toutes les ressources occupées par l'objet SqlDataReader
Read ()	**Permet d'avancer l'objet SqlDataReader jusqu'à l'enregistrement suivant.**
GetDateTime(indicolonne)	Obtient la valeur de la colonne spécifiée sous la forme d'un objet **`DateTime`**.
GetDecimal(indicolonne)	Obtient la valeur de la colonne spécifiée sous la forme d'un objet **`Decimal`**.
GetDouble(indicolonne)	Obtient la valeur de la colonne spécifiée sous la forme d'un nombre de type **`Double`**.
GetFloat(indicolonne)	Obtient la valeur de la colonne spécifiée sous la forme d'un nombre de type **`Float`**.
GetInt16(indicolonne)	Obtient la valeur de la colonne spécifiée sous la forme d'un entier signé 16 bits.
GetInt32(indicolonne)	Obtient la valeur de la colonne spécifiée sous la forme d'un entier signé 32 bits.
GetInt64(indicolonne)	Obtient la valeur de la colonne spécifiée sous la forme d'un entier signé 64 bits.
GetLifetimeService()	Extrait l'objet de service de durée de vie en cours qui contrôle la stratégie de durée de vie de cette instance.
GetName(indicolonne)	Obtient le nom de la colonne spécifiée.
GetString(indicolonne)	Obtient la valeur de la colonne spécifiée sous la forme d'une chaîne.

Par défaut, un **DataReader** charge une ligne entière en mémoire à chaque appel de la méthode **Read()**. Il est possible d'accéder aux valeurs de colonnes soit par leurs noms soit par leurs références ordinales. Une solution plus performante est proposée permettant d'accéder aux valeurs dans leurs types de données natifs (**GetInt64**, **GetDouble**, **GetString**). Par exemple si la première colonne de la ligne indicée par 0 est de type **int**, alors il est possible de la récupérer à l'aide de la méthode **GetInt64** de l'objet **DataReader**.

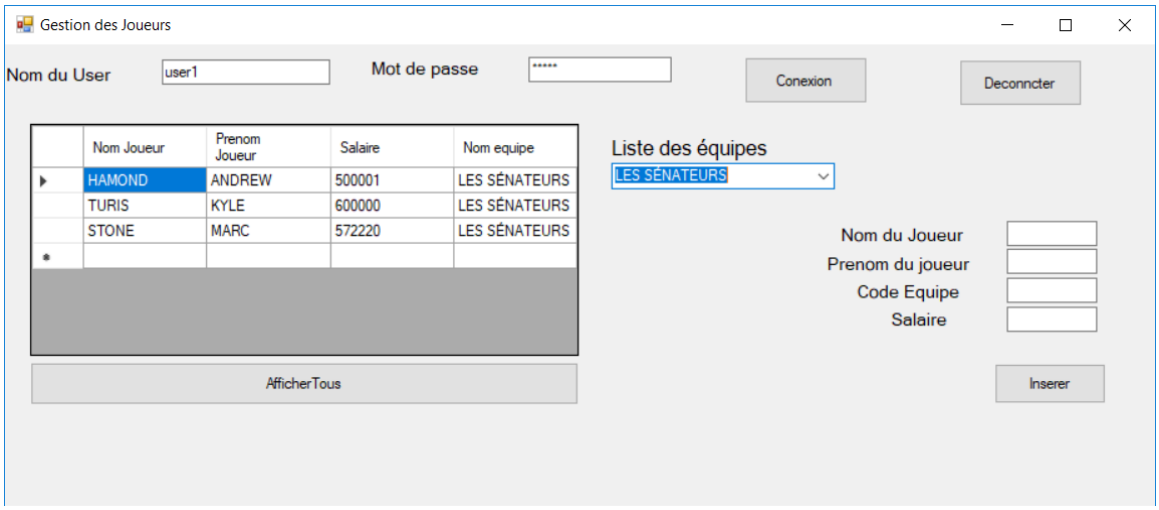
Exemple (on est déjà connecté)

```
string sql2 = "select  nom, prenom from employesClg where typeEmploye = P' ";
SqlCommand sqlCommdSelect = new SqlCommand(sql2, connSql);
 SqlDataReader sqlRead = sqlCommdSelect.ExecuteReader();

          while (sqlRead.Read())
                 {
               Console.WriteLine("{0} - {1}",
               sqlRead.GetString(0),sqlRead.GetString(1));
                }
  sqlRead.Close();
  sqlRead.Dispose();
```

Chapitre 5, exemple avec Windows Form

Présentation

Voici la forme que nous souhaitons avoir.



Sur la forme, il ya les objets

- Un bouton pour se connecter
- Un bouton pour se déconecter
- Un bouton pour afficher tous les joueurs
- Un bouton pour ajouter un joueur.

Chaque bouton appelle la fonction qui lui correspond.

Il ya aussi

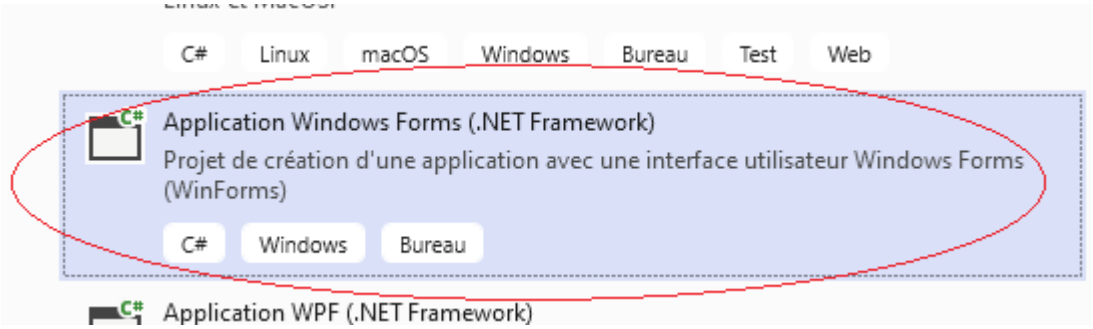
- deux zones de texte pour saiasir le non du user et le mot de passe pou à la base de données et
- 4 zonnes de texte pour rentrer les données du joueur.

Également

- Un Combobox pour contenir la liste des noms des équipes (le nom de l'équipe). Les comboBox peuvent contenir des ITEMS , les **Items** vont correspondre à une colonne de la BD
- Un DataGridView pour contenir la liste de joueurs. Un DataGridView est un objest qui possède des lignes **Rows**. et des colonnes. Même structure qu'une table.

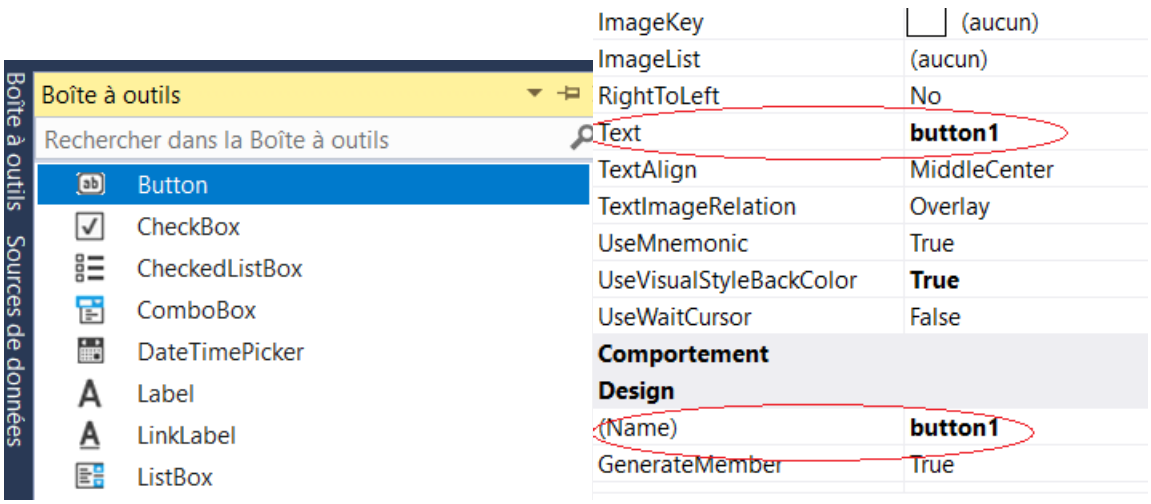
Tous ces objets sont dans une Form Gestion des joueurs.

D'abord créer un projet Windows Form



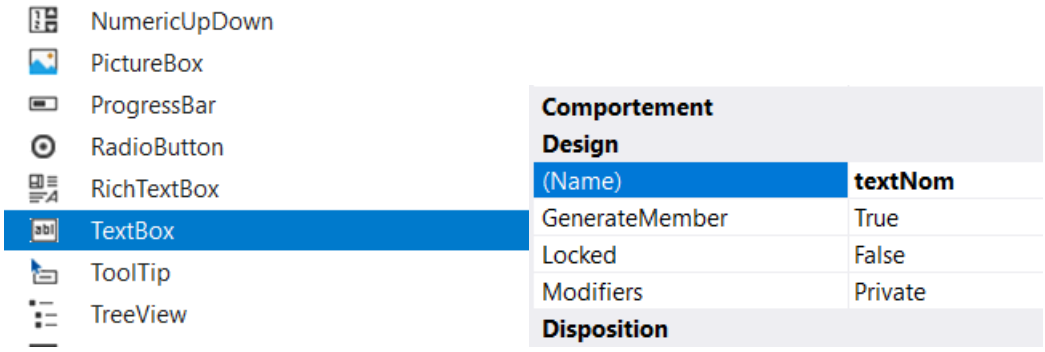
Construction de l'interface. (Attention !! cette méthode n'est pas la meilleure façon de faire)

Une fois que votre projet est démarré, vous trouverez les contrôles dans **la boite à outils.** Voici un aperçu. Il suffit de glisser le contrôle et de le mettre sur la Form



Chaque contrôle a ses propriétés. Quand le contrôle est sélectionné, vous verrez ses propriétés à droite. La propriété la plus importante et le NOM (name), car cette propriété va représenter un nom de variable. Exemple pour le bouton, La propriété Text va représenter ce qui va être afficher sur le bouton. Le name va être le nom bouton dans le code C#

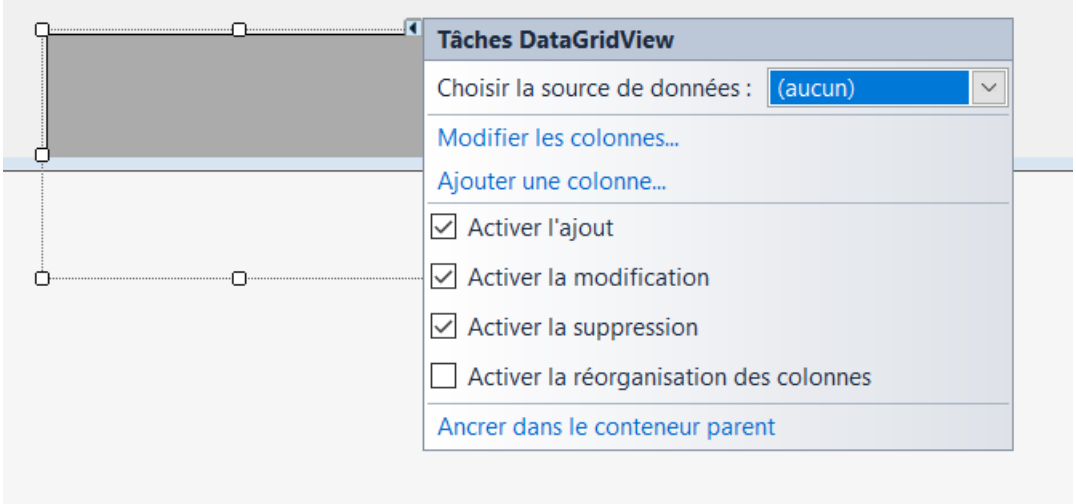
Même principe pour les zones de texte. Le nom est très important



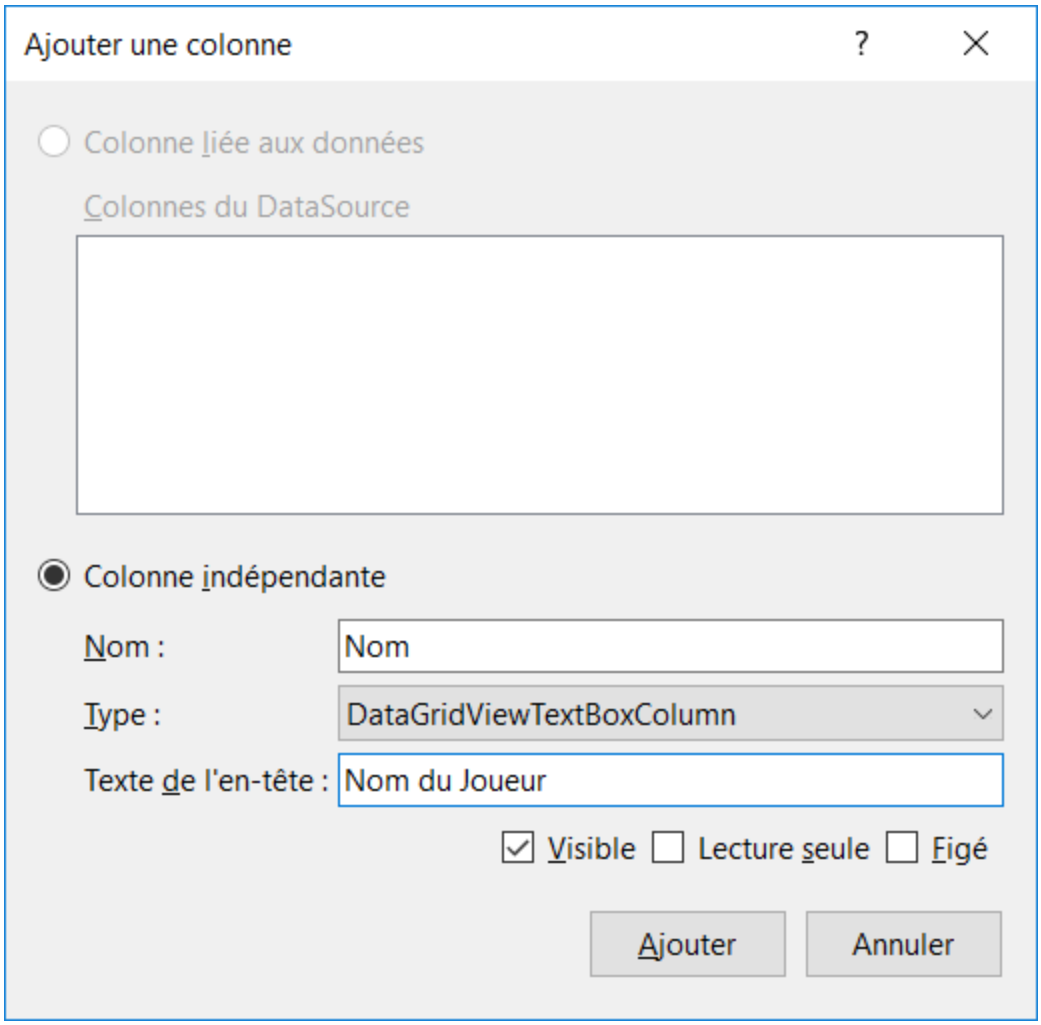
Le DataGridView

Une fois que votre DataGridViw (DGV) est placé sur la form

1- NE CHOISIR aucune source de données.
2- Donner lui un nom significatif
3- Puis cliquer sur ajouter une colonne.

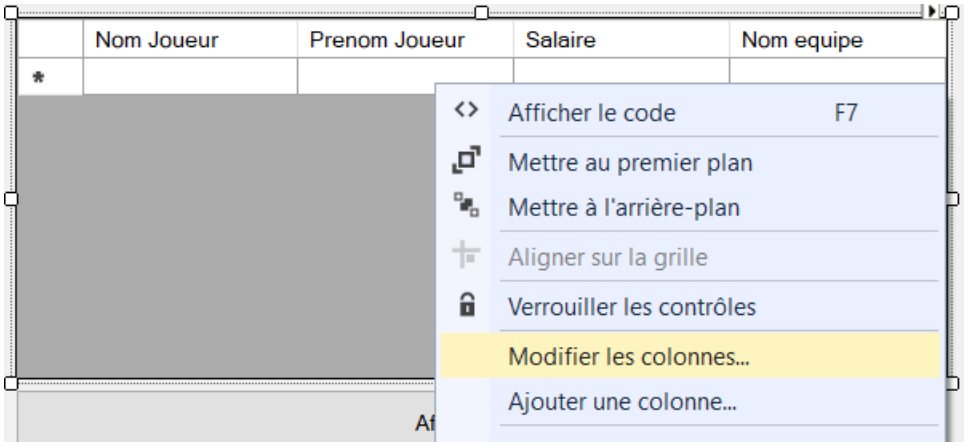


Le nom est le nom de la variable elle-même, texte de l'entête est ce qui va s'afficher sur le DGV

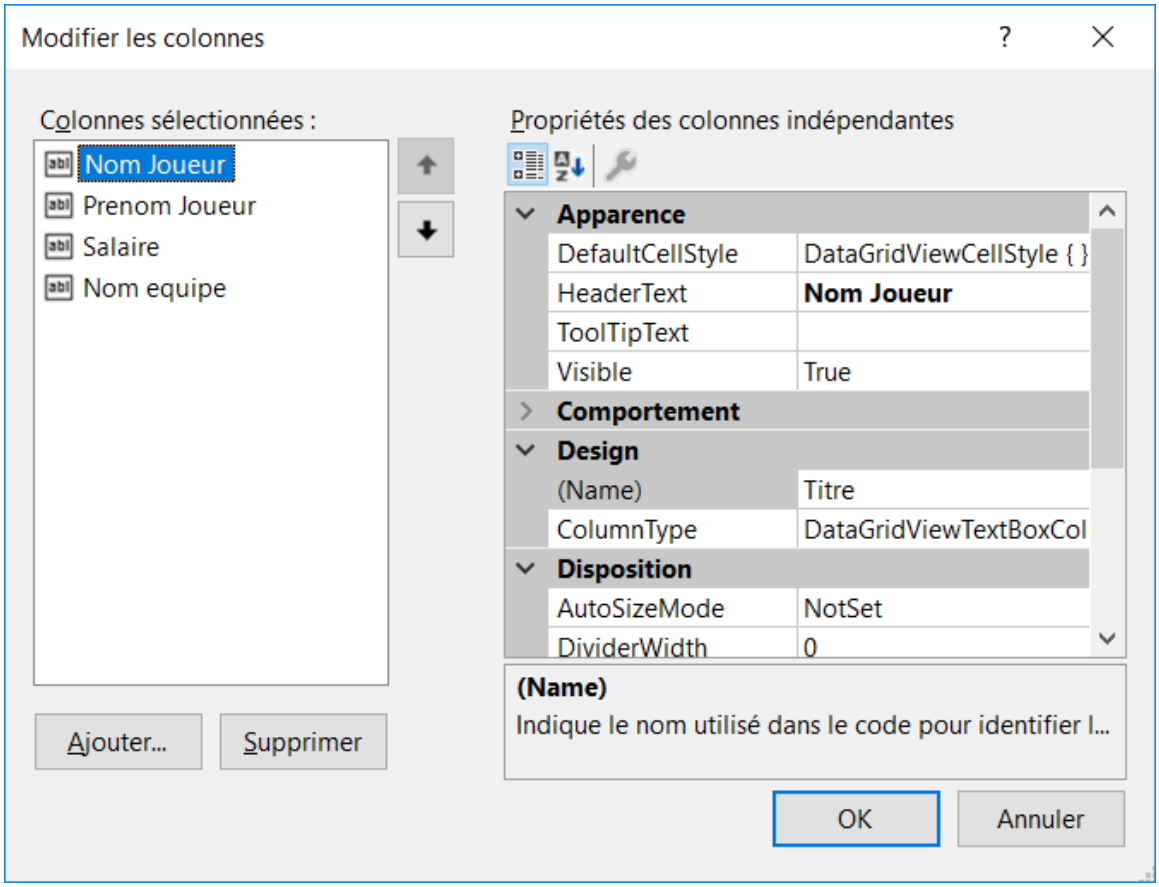


Ajouter toutes les colonnes qui vont correspondre au résultat de votre requête

Note : Quand le DGV est sélectionné, par le bouton droit de la SOURIS, vous pouvez le gérer comme vous voulez.



Si vous avez choisi : Modifier une colonne, vous allez avoir ceci et donc modifier vos colonnes.



Le code C#

Comme nous l'avons mentionné plus chaque bouton va appeler la fonction qui lui correspond.

Pour accéder au code du bouton, il suffit de double cliquer dessus.

Attention : 

Votre objet OracleConnection doit être global, puisque TOUS les objets SqlCommad vont l'utiliser.

```
namespace Kb6Tpno1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

private SqlConnection sqlconn = new SqlConnection();

-------Suite du code
}
}
```

Le bouton btnConnecter

```
private void btnConnecter_Click(object sender, EventArgs e)
        {
            connecter();

            listerEquipe();
        }
```

Voir les explications de `listerEquipe() plus bas.`

Démarche pour la fonction **connecter()**

1- `string user = textId.Text,` on récupère le nom du User par le TextBox de nom TextId. Et on fait la même chose pour le mot de passe.
2- On construit la chaine de connexion
3- On fixe la propriété ConnectionString de l'objet oraconn déclaré plus haut
4- On ouvre une connexion avec Open();
5- Le code est sur la page suivante.

La fonction **connecter()** a le code suivant :

```
        private void connecter()
         {
             string user = textId.Text;
             string passwd = textPassword.Text;
             string chaineConnexion = "Data source =M-INFO-SY\\SQLEXPRESS; Initial
              Catalog  =EmpClgDB; User Id= " + user + ";Password=" + passwd;
             sqlconn.ConnectionString = chaineConnexion;
             sqlconn.Open();
             MessageBox.Show(sqlconn.State.ToString());
         }
```

Toute connexion ouverte doit être fermée. (bouton **Deconnecter**)

```
private void btnDeconnectin_Click(object sender, EventArgs e)
        {
            deconnceter();
        }
```

La fonction deconnecter()

```
private void deconnecter()
        {
            sqlconn.Close();
            MessageBox.Show(sqlconn.State.ToString());
        }
```

Après la connexion, on pourra alors consulter liste des joueurs en utilisant le bouton **Afficher Tous**

Voici le code du bouton **Afficher Tous**

```
private void btnAfficher_Click(object sender, EventArgs e)
        {
            affichertous();
        }
```

La fonction **affichertous()**

```
private void afficherTous()
        {
       dataGridJoueurs.Rows.Clear();
            try
            {
                string sqlselect = "select nom,prenom, salaire, nomEquipe from
Joueurs inner join equipes on Equipes.codeEquipe = Joueurs.codeEquipe";
                SqlCommand sqlCmd = new SqlCommand(sqlselect, sqlconn);
                SqlDataReader sqlReader = sqlCmd.ExecuteReader();
                    while (sqlReader.Read())
                     {
                    dataGridJoueurs.Rows.Add(sqlReader.GetString(0),
                    sqlReader.GetString(1), sqlReader.GetDecimal(2),
                    sqlReader.GetString(3));
                     }
                sqlReader.Close();
            }

            catch (Exception exsql1)
            {
                MessageBox.Show(exsql1.Message.ToString());
            }
        }
```

Explications, la fonction affichertous()

dataGridJoueurs.**Rows.Clear();** permet de vider le DGV, sinon il va se remplir encore et encore à chaque clique du bouton

string sqlselect = "select nom, …… " on construit la requêtes SQL. On construit la requête dans le try pour pouvoir vérifier sa validité.

```
SqlCommand sqlCmd = new SqlCommand(sqlselect, sqlconn);
```

On crée un Obejt SqlCommad en lui passant la requete sqlselect et la connexion sqlconn.

```
SqlDataReader sqlReader = sqlCmd.ExecuteReader();
```

Comme c'Est une requête SELECT, on utilise la méthode ExecuteReader() de l'objet SqlCommand. Ce qui permet d'obtenir un objet SqlDataReader.

`while (sqlReader.Read())` On lit l'objet SqlDatareader ligne par ligne en utilisant la méthode Read();

`sqlReader`**.GetString(0),** On utilise la méthode getTypedeDonnee(Indicedecolonne) pour lire le contenu d'une colonne.

```
            while (sqlReader.Read())
                   {
                  dataGridJoueurs.Rows.Add(sqlReader.GetString(0),
                  sqlReader.GetString(1), sqlReader.GetDecimal(2),
                  sqlReader.GetString(3));
                   }
```

La boucle while qui permet de lire un DataReader ligne par ligne, va permettre de remplir le DGV ligne par ligne aussi. On utilise la propriété **Rows** et la méthode **Add()** pour ajouter une ligne à chaque fois.

```
sqlReader.Close();Il faut fermer l'objet SqlDataReader avec sa méthode Close();
```

La fonction listeEquipe(), pour afficher la liste des équipes dans un ComboBox ou dans une liste (pas un DGV)

La fonction qui correspond à afficher la liste des équipes dans un ComboBox est la suivante :

```
private void listerEquipe()
        {

            string sqlequipe = "Select  nomequipe from equipes";
            SqlCommand sqlcmd2 = new SqlCommand(sqlequipe, sqlconn);
            SqlDataReader sqlRedEquipe = sqlcmd2.ExecuteReader();
            while (sqlRedEquipe.Read())
                {
                comEquipes.Items.Add(sqlRedEquipe.GetString(0));
                }
           sqlRedEquipe.Close();
           comEquipes.SelectedIndex = 0;


        }
```

Explications de la fonction `listerEquipe()`

On liste les équipes et on les affiche dans un ComboBox. Le nom du ComboBox est comEquipes

Vous l'avez compris. Tout le code avant le while s'explique de la même façon que pour la fonction affichertous();

Vous avez compris aussi que la requête SELECT dans ce cas renvoi une seule colonne. Dans ce cas on va utiliser un ComboBox pour contenir le résultat de la requête.

Pour le ComboBox on va utiliser la propriété **Items** et la méthode **Add()**

```
comEquipes.Items.Add(sqlRedEquipe.GetString(0));
```

Va permettre d'ajouter un Items(une colonne) à notre ComboBox. Évidement cette instruction est dans un while puisqu'il faut lire tout le SqlDataReader SqlReadEquipe.

comEquipes.SelectedIndex = 0; va positionner l'index de la ComboBox à zéro, donc va pouvoir afficher le premier Items (ou la première ligne de la requête).

Attention : 

La fonction `listerEquipe()` doit être appelée. Ici nous avons décidé de l'appeler juste après la connexion.

Et si on veut afficher la liste des joueurs d'une équipe donnée ?

Le nom de l'équipe est déjà dans le comboBox, il suffit de faire une requête avec un WHERE sur le contenu du ComboBox.

Voici la fonction qui liste les joueurs selon leur équipe.

Fonction `listejoueursEquipe()`

```
private void listejoueursEquipe()
        {
            string nomEquip = comEquipes.Text;
            try
            {
                string sql2 = "select nom, prenom,salaire, nomequipe from joueurs
                inner join equipes on joueurs.CODEEQUIPE = equipes.CODEEQUIPE
                where nomEquipe ='" + nomEquip + "'";

                SqlCommand sqlcmd3 = new SqlCommand(sql2, sqlconn);
                SqlDataReader sqlReadJoueur = sqlcmd3.ExecuteReader();

                while (sqlReadJoueur.Read())

                     {
                    dataGridJoueurs.Rows.Add(sqlReadJoueur.GetString(0),
                        sqlReadJoueur.GetString(1),
                        sqlReadJoueur.GetDecimal(2),
                        sqlReadJoueur.GetString(3));
                    }
                sqlReadJoueur.Close();

            }
            catch (Exception exsql2)
             {
                MessageBox.Show(exsql2.Message.ToString());
             }
         }
```

Explications de la fonction `listejoueursEquipe()`

On utilise une requête avec jointure, puisque la clause WHERE porte sur le nom de l'équipe.

- La requête est SELECT nom,prenom, salaire ……Cette requête renvoi plusieurs lignes et plusieurs colonnes. Donc pour afficher le résultat on utilise un Un DataGridView.(voir plus haut).
- La requête présente une clause WHERE sur le nom de l'équipe. Or le nom de l'équipe est dans le comboBox de nom comEquipes. Il suffit de le récupérer comme suit :
  `string nomEquip = comEquipes.Text;`

  La requête est donc construite ainsi:

```
string sql2 = "select nom, prenom,salaire, nomequipe   from joueurs inner join
equipes on joueurs.CODEEQUIPE = equipes.CODEEQUIPE where nomEquipe ='" + nomEquip
+ "'";
```

Une fois la requête construite, il suffit d'utiliser un SqlCommand pour l'envoyer au SGBD et appliquer la méthode ExecuteReader() pour obtenir le résultat dans un SqlDataReader.

On lit le SqlDataReader avec la méthode Read() et une boucle while. Il suffit d'envoyer le résultat de la lecture dans le DataGridView. On ajoute une ligne à la fois avec Rows.Add(..)

```
while (sqlReadJoueur.Read())

                    {
                       dataGridJoueurs.Rows.Add(sqlReadJoueur.GetString(0),
                       sqlReadJoueur.GetString(1),
                       sqlReadJoueur.GetDecimal(2),
                       sqlReadJoueur.GetString(3));
                    }
```

Cette fonction sera appelée lorsqu'on choisit (on change) le nom de l'équipe dans le comboBox. Le comboBox a un évènement SelectedIndexChanged `(lorsque l'index est changé)`

Code du combBox :

```
private void comEquipes_SelectedIndexChanged(object sender, EventArgs e)
            {
                dataGridJoueurs.Rows.Clear();
                listejoueursEquipe();

            }
```

`dataGridJoueurs.Rows.Clear();` permet de vider le DataGridView. Sinon il va se remplir tout le temps.

Attention : 

Il faut effacer le contenu de votre DataGridView si vous ne voulez pas qu'il se rempli à l'infini

Insertion des données dans la base de données.

L'insertion des données va se faire par les zones de texte `texNom`, `textPrenom`, `textSalaire` et `textCodeEquipe`. Donc la première chose à faire est de récupérer leur contenu, et les mettre dans des variables :

```
            string nomjoueur = textNom.Text;
            string prnjoueur = textPrenom.Text;
            string salairejoueur = textSalaire.Text;
            string codeEquipe = textCodequipe.Text
```

Attention : 

1- Le numéro du joueur est inséré automatiquement par IDENTITY
2- Remarquez que les simples guillemets lorsque la colonne concernée par la saisie est de type VARCHAR (donc un string)

Il suffit de construire la requête INSERT INTO ..

```
        private void inserer()
        {
            string nomjoueur = textNom.Text;
            string prnjoueur = textPrenom.Text;
            string salairejoueur = textSalaire.Text;
            string codeEquipe = textCodequipe.Text;
            string sqlinsert = " insert into joueurs(nom, prenom,codeequipe,
       salaire) values ('" + nomjoueur + "','" + prnjoueur + "','" + codeEquipe +
              "'," + salairejoueur + ")";
            SqlCommand sqlcmdInsert = new SqlCommand(sqlinsert,sqlconn);
            sqlcmdInsert.ExecuteNonQuery();
            vider();
        }
```

Une fois que la requête est construite, on utilise un SqlCommand avec sa méthode ExecuteNonQuery() pour envoyer et exécuter la requête. Rien de plus simple.

Comme, les zones de textes seront utilisées pour effectuer plusieurs insertions, il est conseillé de les vider pour la prochaine insertion.

```
private void vider()
        {
            texNom.Clear();
            textPrenom.Clear();
            textCodeEquipe.Clear();
            textSalaire.Clear();
        }
```

La fonction est appelée par le bouton btnInserer comme suit.

```
private void btnInserer_Click(object sender, EventArgs e)
        {
            inserer();
            vider();
        }
```

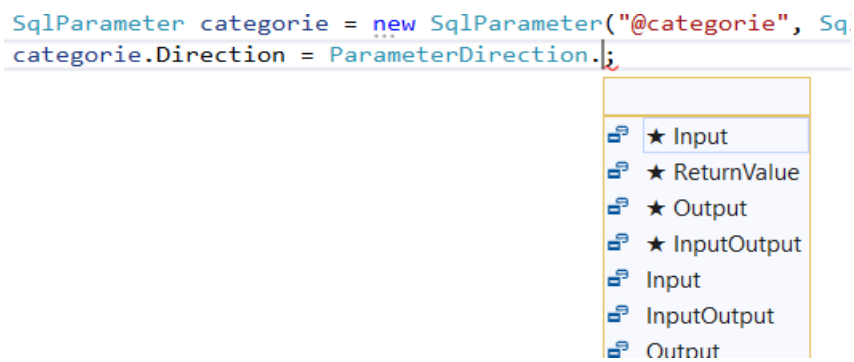
Chapitre 6, l'objet SqlParameter : Procédures stockées

L'objet SqlParameter représente un paramètre pour l'SqlCommand ou une colonne du DataSet

Quelques Constructeurs

SqlParameter()	Instancie un SqlParameter
SqlParameter (string, SqlDbType)	String désigne le nom du paramètre, le SqlDbtype désigne le type de données du Pramètre (un SqlType.) : public SqlParameter(string parameterName, SqlDbType oraType)
SqlParameter(string, SqlDbType, int)	Même que le précédent, sauf qu'on indique la taille du paramètre
SqlParameter(string, SqlDbType, int, string)	Même que le précédent, sauf qu'on indique la taille du paramètre et le nom de la colonne source : à voir avec le DataSet
SqlParameter(string,SqlDbType,ParameterDirection)	Le ParameterDirection indique si le paramètre est en IN ou Out. **Utilisé lorsque nous avons une procédure stockée**

La direction des paramètres peut être :



Input, lorsque les paramètres de la procédure sont en IN

OutPut lorsque les paramètres de la procédure sont en OUT

ReturnValue lorsqu'il s'agit d'une fonction

Propriétés importantes SqlParameter

Direction	Obtient ou définit une valeur qui indique si le paramètre est un paramètre d'entrée uniquement, de sortie uniquement, bidirectionnel ou de valeur de retour d'une procédure stockée.
ParameterName	Obtient ou définit le nom de SqlParameter
SqlDbType	Spécifie le type de données Sql

Size	Obtient ou définit la taille maximale, en octets, des données figurant dans la colonne.
Value	Obtient ou définit la valeur du paramètre

Méthodes importantes de l'objet OrcaleParameter

Colone()	Crée une copie de l'objet SqlParameter
Dispose ()	Libère les ressources occupées par SqlParameter
GetType()	Obtient le Type de l'instance actuelle
ToString()	Obtient une chaîne qui contient ParameterName

Pour résumer : (pour l'instant)

1. On se connecte à la base de données avec SqlConnection();
2. Seul SqlCommand permet de passer votre requête ou votre procédure stockée à la base de données.
3. La propriété CommandType de SqlCommnd nous renseigne sur le type de requête (Simple ou procédure stockée). Il sera `CommandType.StoredProcedure;`
4. La propriété `CommandText de SqlCommande indique le nom de la procédure`
5. Les paramètres de votre requête ou de votre procédure stockée sont des objets SqlParameter. Pour chaque paramètre de la procédure vous devez définir un SqLParameter.
6. Pour chaque SqlParameter vous devez définir : Son type, sa direction , sa valeur.
7. Avant d'exécuter votre v procédure, vous devez ajouter la définition des paramètres à votre SqlCommand
8. Exécuter la commande avec ExecuteNonQuery() ou ExecuteReader();

Exemple 1 *:* **Insertion dans la base de données : TOUS les paramètres sont en IN**

On se connecte d'abord à la base de données :

```
        private void connecter()
        {
            try
            {     string dSource = "Data Source=M-INFO-SY\\SQLEXPRESS2017;Initial
Catalog=BdJeu;User ID=patoche;Password=123456";
                conn.ConnectionString = dSource;
                conn.Open();
                MessageBox.Show(conn.State.ToString());
            }
            catch (Exception ex)
            { MessageBox.Show(ex.Message); }
        }
```

Voici le code de la procédure InsertQuestion

```
create procedure InsertQuestion

(@enonce varchar(200), @flag int,@difficulte varchar(20),@categ int) as

begin

insert into Questions values(@enonce,@flag,@difficulte,@categ);

end;
```

Voici le code C# qui appelle la procédure :

La procédure a que des paramètres en IN (implicite) alors dans le code suivant, la direction des paramètres sera **`Input`**

```
        private void inserer()
        {
            try
            {

          //on prépare la requête, qui dans notre cas une procédure stockée
          //remarquez : la cahaine dans SqlCommand indique le nom de la procédure
          //remarquez : CommandText a le nom de la procédure
          //remarquez : CommandType dit que c'est une procédure.

                SqlCommand sqlInsert = new SqlCommand("InsertQuestion", conn);
                sqlInsert.CommandText = "InsertQuestion";
                sqlInsert.CommandType = CommandType.StoredProcedure;

       //declaration des paramètres

          SqlParameter enonce = new SqlParameter("@enonce", SqlDbType.VarChar, 30);
                enonce.Direction = ParameterDirection.Input;

                SqlParameter flag = new SqlParameter("@flag", SqlDbType.Int, 4);
                flag.Direction = ParameterDirection.Input;

      SqlParameter difficulte = new SqlParameter("@difficulte", SqlDbType.VarChar, 30);
                difficulte.Direction = ParameterDirection.Input;

            SqlParameter categorie = new SqlParameter("@categ", SqlDbType.VarChar, 30);
                categorie.Direction = ParameterDirection.Input;

                //affectation des valeurs aux paramètres

                enonce.Value = textEnonce.Text;
                flag.Value = textFalg.Text;
                difficulte.Value = textDifficulte.Text;
                categorie.Value = textCategorie.Text;
```

```
    //L'objet sqlCommand doit passer la requête en tenant compte des paramètres.
            //on utilise la propriété Parameters de l'objet sqlCommand sqlInsert
              sqlInsert.Parameters.Add(enonce);
              sqlInsert.Parameters.Add(flag);
              sqlInsert.Parameters.Add(difficulte);
              sqlInsert.Parameters.Add(categorie);
              sqlInsert.ExecuteNonQuery();

          }
          catch (Exception ex1)
          { MessageBox.Show(ex1.Message); }
      }
```

Exemple 2 : Appel de procédure stockée sans paramètres (ni en IN ni en OUT)

Voici le code de la procédure : Cette procédure affiche les noms de catégories.

```
CREATE procedure [dbo].[listeCategorie] as
begin
select non_ctegorie from categories;
end;
```

Comme la procédure n'a pas de paramètres alors aucun paramètre à définir dans le code ADO.NET . Mais, vous devez préciser qu'il est question d'une procédure stockée.

```
      private void listeCategorie()

      {
          try {
          SqlCommand sqlCategorie = new SqlCommand("listeCategorie", conn);
          sqlCategorie.CommandText = "listeCategorie";
          sqlCategorie.CommandType = CommandType.StoredProcedure;


          SqlDataReader resutat = sqlCategorie.ExecuteReader();
          while (resutat.Read())
               {
                listCategorie.Items.Add(resutat.GetString(0));
               }
          resutat.Close();
          listCategorie.SelectedIndex = 0;
              }
          catch (Exception ex4)
          { MessageBox.Show(ex4.Message); }
      }
```

Exemple 3, Appel de procédure SELECT avec un paramètre en IN

```
CREATE procedure [dbo].[listeQuestion] (@categorie varchar(30))
as
begin
select enonce,difficulte from Questions inner join categories
on Questions.code_categorie = categories.code_Categorie
where non_Ctegorie =@categorie;
end;
```

En principe, rendu ici vous avez compris qu'il faut juste passer le paramètre en IN

```
        private void listQuestions()
        {
            dgvQuestions.Rows.Clear();
            try
            {
                SqlCommand sqlQuestion = new SqlCommand("listeQuestion", conn);
                sqlQuestion.CommandText = "listeQuestion";
                sqlQuestion.CommandType = CommandType.StoredProcedure;

   SqlParameter categorie = new SqlParameter("@categorie", SqlDbType.VarChar, 30);
                categorie.Direction = ParameterDirection.Input;
                categorie.Value = listCategorie.Text;

                sqlQuestion.Parameters.Add(categorie);
                SqlDataReader lisQuestion = sqlQuestion.ExecuteReader();

                while (lisQuestion.Read())

                {
        dgvQuestions.Rows.Add(lisQuestion.GetString(0), lisQuestion.GetString(1));
                }
                lisQuestion.Close();
            }


            catch (Exception ex3)
            { MessageBox.Show(ex3.Message); }
       }
```

Exemple 4, appel d'une fonction stockée qui retourne un scalaire

La fonction suivante retourne le nombre total de Question

```
create function compterQuestion() returns int
 as
 begin
 declare @total int;
 select @total = count(*) from Questions;
 return @total;
 end;
```

la fonction sera passée de la même façons qu'une procédure en indiquant la direction du paramètre de retour de la fonction. Dans ce cas c'Est un **`ReturnValue`**

```
       private void compterQuestion()
        {
            try
            {
                SqlCommand sqlTotal = new SqlCommand("compterQuestion", conn);
                sqlTotal.CommandText = "compterQuestion";
                sqlTotal.CommandType = CommandType.StoredProcedure;


                SqlParameter resultat = new SqlParameter("@total", SqlDbType.Int);
                resultat.Direction = ParameterDirection.ReturnValue;

                sqlTotal.Parameters.Add(resultat);
                sqlTotal.ExecuteScalar();
                textTotal.Text = resultat.Value.ToString();


            }

            catch (Exception ex6)
            { MessageBox.Show(ex6.Message); }
        }
```

Exemple 5, appel d'une fonction stockée qui retourne une table

Dans cet exemple, on retombe dans du code C# sans procédure stockées. C'est comme si nous avons une simple requête paramétrée, la raison est que l'exécution d'une fonction table se fait comme suit dans SQL Server Management Studio

```
select * from chercherQuestion('Art'); Art, étant la valeur du paramètre
```

```
Create FUNCTION chercherQuestion(@categorie varchar(30))
  returns table
 AS
 return(
 select enonce,difficulte from Questions inner join categories
on Questions.code_categorie = categories.code_Categorie where non_Ctegorie
=@categorie);
```

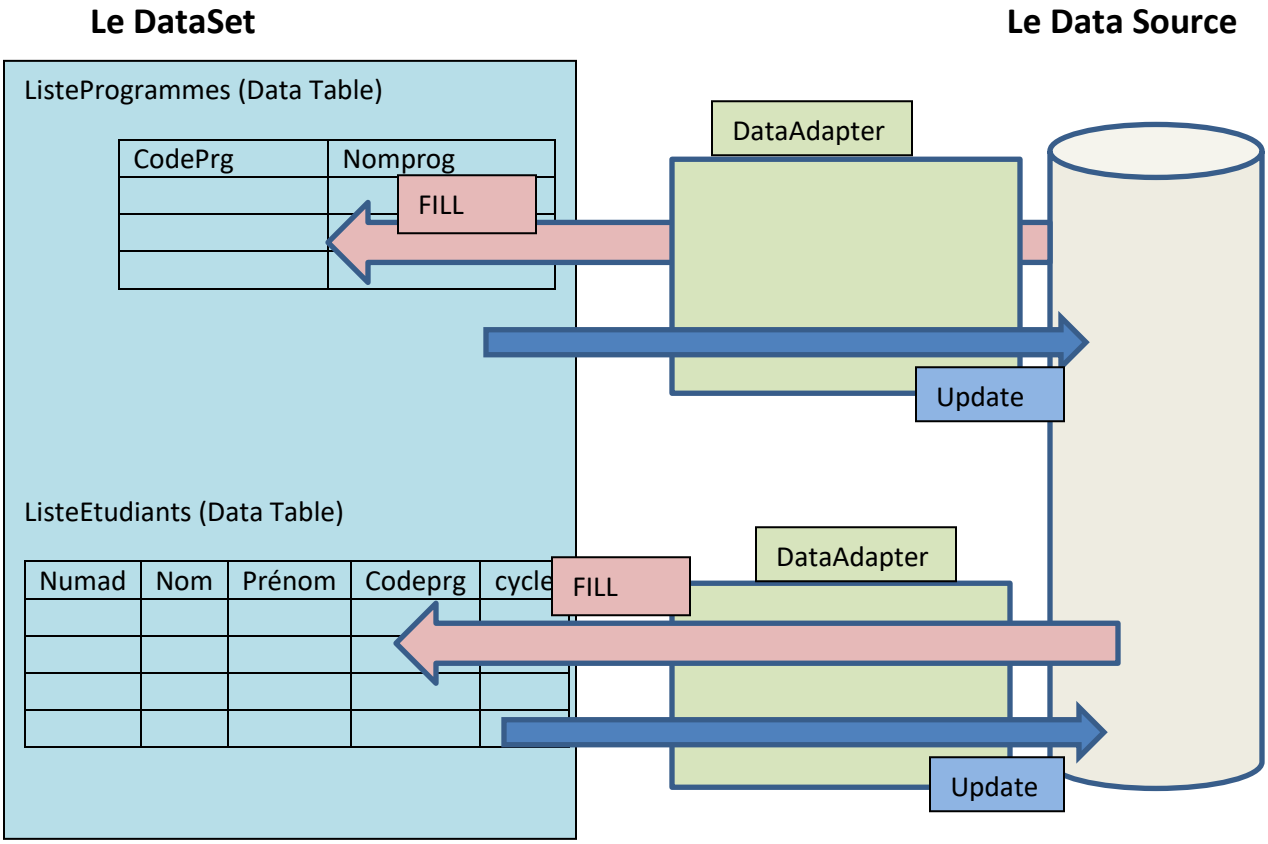
Code ADO.NET

```
private void listQuestionsFunction()
        {
            dgvQuestions.Rows.Clear();
            try
            {
                string sql1= "select * from chercherQuestion(@categorie)";


                SqlCommand sqlQuestion2 = new SqlCommand(sql1, conn);
               sqlQuestion2.CommandText = sql1;
                sqlQuestion2.CommandType = CommandType.Text;

   SqlParameter categorie = new SqlParameter("@categorie", SqlDbType.VarChar, 30);
                categorie.Value = listCategorie.Text;

                sqlQuestion2.Parameters.Add(categorie);
                SqlDataReader resQuestion = sqlQuestion2.ExecuteReader();

                while (resQuestion.Read())

                {
        dgvQuestions.Rows.Add(resQuestion.GetString(0), resQuestion.GetString(1));
                }
                resQuestion.Close();
            }


            catch (Exception ex5)
            { MessageBox.Show(ex5.Message); }
        }
```

Chapitre 7, Le DataSet (Ne rentre pas dans le cours)

L'objet DataSet d'ADO.NET est une représentation résidente en mémoire de données, qui propose un modèle de programmation relationnel cohérent, indépendant de la source de données. Le DataSet représente un jeu de données complet qui comprend des tables, des contraintes et des relations entre les tables. Étant donné que le DataSet est indépendant de la source de données, le DataSet peut inclure des données locales par rapport à l'application ainsi que des données provenant de plusieurs sources. L'interaction avec les sources de données existantes est contrôlée par le DataAdapter.

Il est possible d'utiliser un nombre quelconque d'objets DataAdapter avec un DataSet. Chaque DataAdapter peut être utilisé pour remplir un ou plusieurs objets DataTable et répercuter les mises à jour dans la source de données concernée. Les objets DataRelation et Constraint peuvent être ajoutés localement au DataSet, ce qui vous permet de relier des données provenant de sources de données hétérogènes. Par exemple, un DataSet peut contenir des données provenant d'une base de données Microsoft SQL Server, d'une base de données IBM DB2 exposée via OLE DB et d'une source de données qui diffuse le XML en continu. Un ou plusieurs objets DataAdapter peuvent gérer la communication vers chaque source de données.



Propriétés importantes de l'objet DataSet

Relations	Obtient la collection des relations qui relient des tables et permettent de naviguer des tables parentes aux tables enfants.
Tables	Obtient la collection des tables contenues dans **`DataSet`**.
CaseSensitive	Obtient ou définit une valeur indiquant si les comparaisons de chaînes au sein d'objets **`DataTable`** respectent la casse.
DataSetName	Obtient ou définit le nom du **`DataSet`** en cours.
EnforceConstraints	Obtient ou définit une valeur indiquant si les règles de contrainte doivent être respectées lorsque vous tentez une opération de mise à jour.
HasErrors	Obtient une valeur indiquant s'il existe des erreurs dans les objets **`DataTable`** de ce **`DataSet`**.
Locale	Obtient ou définit les paramètres régionaux utilisés pour comparer des chaînes dans la table.
Namespace	Obtient ou définit l'espace de noms de **`DataSet`**.
Prefix	Obtient ou définit un préfixe XML qui associe un alias à l'espace de noms de **`DataSet`**.

Méthodes importantes de l'objet DataSet

AcceptChanges()	Valide toutes les modifications apportées à ce **`DataSet`** depuis son chargement ou depuis le dernier appel à **`AcceptChanges`**.
Clear()	Efface toutes les données de **`DataSet`** en supprimant toutes les lignes de l'ensemble des tables.
Clone()	Copie la structure de **`DataSet`**, y compris tous les schémas, relations et contraintes **`DataTable`**. Ne copie aucune donnée.
Copy()	Copie à la fois la structure et les données de ce **`DataSet`**.

GetChanges()	Obtient une copie du **DataSet** contenant l'ensemble des modifications qui lui ont été apportées depuis son dernier chargement ou depuis l'appel à **AcceptChanges**.
GetXml()	Retourne la représentation XML des données stockées dans **DataSet**.
GetXmlSchema()	Retourne le schéma XSD de la représentation XML des données stockées dans **DataSet**.
HasChanges	Obtient une valeur indiquant si **DataSet** contient des modifications, notamment des lignes nouvelles, supprimées ou modifiées.
Merge	Fusionne un **DataSet**, un **DataTable** ou un tableau d'objets **DataRow** dans le **DataSet** ou le **DataTable** en cours.
RejectChanges	Annule toutes les modifications apportées à **DataSet** depuis sa création ou le dernier appel à **DataSet.AcceptChanges**.
Reset	Rétablit l'état d'origine de **DataSet**. Les sous-classes doivent substituer Reset pour rétablir l'état d'origine de **DataSet**.

http://msdn.microsoft.com/fr-fr/library/System.Data.DataSet%28v=vs.110%29.aspx

L'objet SqlDataAdapter

L'objet **OracleDataAdapter** fonctionne comme un pont entre le **DataSet** et les données source. Il permet de peupler le DataSet Par les données issues de la source de données (SELECT) et de mettre à jour la base de données par les données du DataSet.

L'objet **OracleDataAdapter** utilise des commandes pour mettre à jour la source de données après que des modifications aient été apportées au **DataSet**. Quand elle est utilisée, la méthode **Fill** de l'objet **OracleDataAdapter** appelle la commande **SELECT**, en utilisant la méthode **Update**, appelle la commande **INSERT**, **UPDATE** ou **DELETE** pour chaque ligne modifiée. Vous pouvez explicitement définir ces commandes afin de contrôler les instructions utilisées au moment de l'exécution pour résoudre les modifications, y compris en utilisant des procédures stockées.

Quelques constructeurs de l'objet SqlDataAdpter

SqlDataAdapter()	Instancie un objet SqlDataAdapter
SqlDataAdapter(String, SqlConnection)	Instancie un objet un objet SqlConnection avec la commande SQL SELECT et une connexion à la BD.(Ici, le SELECT est passé par le SqlDataAdapter)
SqlDataAdapter(SqlCommand)	Initialise une nouvelle instance de la classe SqlDataAdapter avec l'instruction SQL SELECT spécifiée. (Ici, le SELECT est contenu dans le SqlCommand)

Propriétés importantes de l'objet SqlDataAdpter

AcceptChangesDuringFill	Obtient ou définit une valeur indiquant si **`AcceptChanges`** est appelé sur **`DataRow`** après son ajout à **`DataTable`** durant les opérations **`Fill`**.
ContinueUpdateOnError	Obtient ou définit une valeur qui spécifie si une exception doit être générée en cas d'erreur pendant la mise à jour d'une ligne.
DeleteCommand	Obtient ou définit une instruction SQL ou une procédure stockée utilisée pour supprimer des enregistrements dans la base de données.
InsertCommand	Obtient ou définit une instruction SQL ou une procédure stockée utilisée pour insérer de nouveaux enregistrements dans la base de données.
SelectCommand	Obtient ou définit une instruction SQL ou une procédure stockée utilisée pour sélectionner des enregistrements dans la base de données.
UpdateCommand	Obtient ou définit une instruction SQL ou une procédure stockée utilisée pour mettre à jour des enregistrements dans la base de données

Méthodes importantes de l'objet SqlDataAdpter

Fill()	Ajoute ou actualise des lignes de **`DataSet`** pour qu'elles correspondent à celles de la source de données.
Dispose()	Libère les ressources utilisées par **`SqlDataAdpter`**
Update()	Appelle les instructions INSERT, UPDATE ou DELETE respectives pour chaque ligne insérée, mise à jour ou supprimée dans **`DataSet`**.

Événements importants de l'objet SqlDataAdpter

FillError	Retourné lorsqu'une erreur se produit pendant une opération de remplissage.
RowUpdated	Se produit lors d'une opération de mise à jour après l'exécution d'une commande sur la base de données.
RowUpdating	Se produit pendant une opération de Update, avant l'exécution d'une commande sur la source de données.