

# ACCES AUX BASES DE DONNEES

- **Najlae IDRISI**
- Université Sultan Moulay Slimane
- Faculté des Sciences et Techniques
- Béni Mellal
- Département Informatique
- © version 2024/2025

# Plan

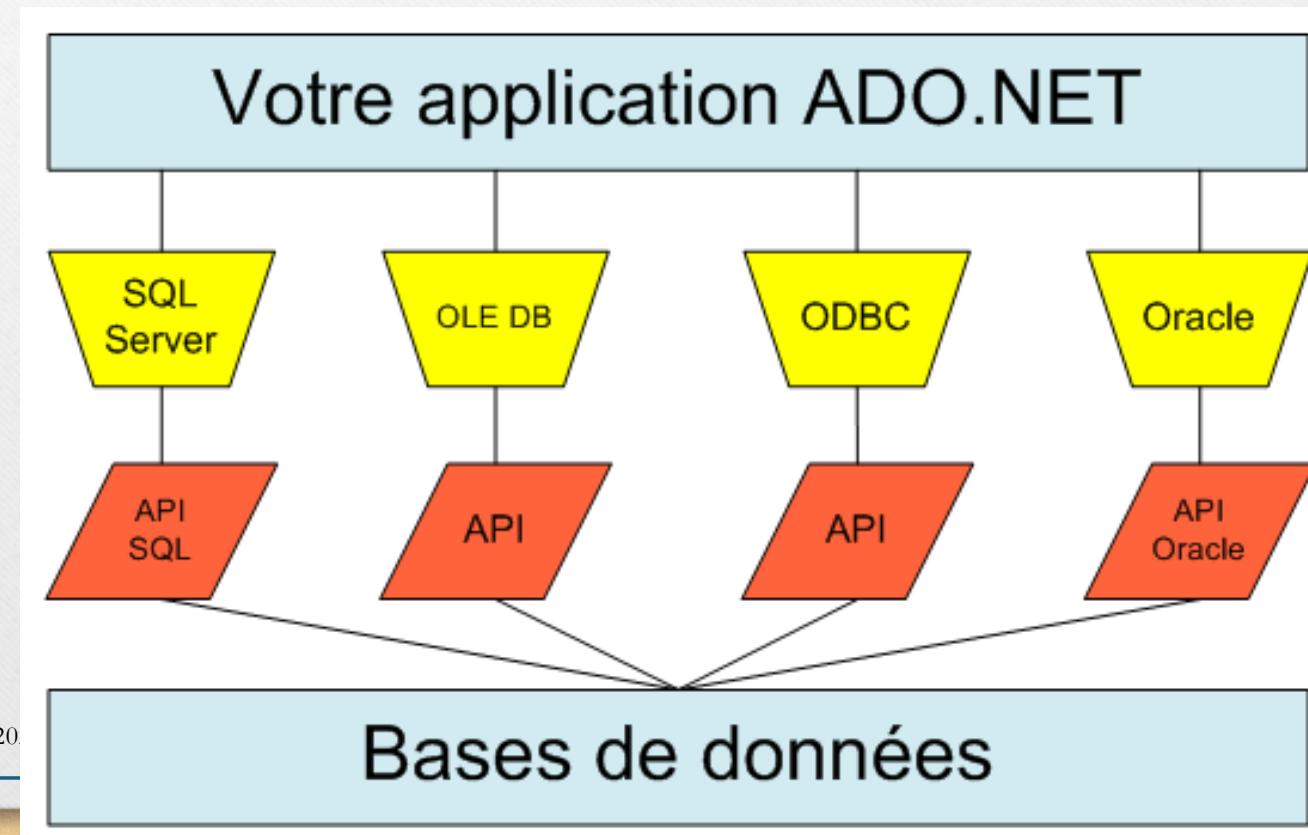
- ADO.NET
- Bases de données
- Mode connecté
- Mode déconnecté
- Exemples

# Qu'est ce que l'ADO.NET ?

- ADO.NET (*ActiveX Data Objects.Net*) est un ensemble de classes d'accès aux données compris dans le Framework .NET 3.5. Il permet de gérer de façon simplifiée et organisée des données stockées en base (relationnelle ou non), dans des fichiers XML (*eXtensible Markup Language*) et API (*Application Programming Interface*).
- ADO.NET permet au développeur de faciliter la manipulation des données en établissant la liaison entre les données et l'application. Il existe 2 modes de connexion : connecté et déconnecté.
  - *Le mode connecté donne un accès en permanence aux données donc une synchronisation quasi immédiate mais demande une bonne prise en charge du réseau.*
  - *Le mode déconnecté permet de stocker temporairement le résultat de nos requêtes en se connectant juste le temps de la récupération, de les modifier et de mettre à jour la base en se reconnectant.*
- ADO.NET nous offre 4 fournisseurs pour accéder aux données (SQL Server, Oracle, ODBC et OLE DB). Ceux-ci peuvent permettre de stocker dans un DataSet les résultats de nos requêtes.

# Fournisseurs de données

- Chaque fournisseur de données permet la communication avec un type de base de données au travers d'une API. Une API est l'interface qui permet l'accès de logiciel par un autre. Ces fournisseurs permettent de récupérer et de transférer des modifications entre l'application et une base de données. Toutes les classes permettant d'utiliser des fournisseurs se trouvent dans l'espace de nom System.Data.
- Sur le Framework 3.5, il existe quatre types de fournisseurs :
  - Sql Server
  - OLE DB
  - OBDC
  - Oracle



- Chaque fournisseur est relié à une base de données propre, c'est-à-dire qu'il est compatible à l'API de sa base de données. Cependant, 2 sont spécifiques : SQL Server pour les bases SQL Server et Oracle pour les bases Oracle. Les 2 autres sont plutôt génériques

| Fournisseur | Description   |
|-------------|---|
| SQL Server  | Les classes de ce fournisseur se trouvent dans l'espace de nom <code>System.Data.SqlClient</code> , chaque nom de ces classes est préfixé par <code>Sql</code> . SQL Server à accès au serveur sans utiliser d'autres couches logicielles le rendant plus performant.   |
| OLE DB      | Les classes de ce fournisseur se trouvent dans l'espace de nom <code>System.Data.OleDb</code> , chaque nom de ces classes est préfixé par <code>OleDb</code> . Ce fournisseur exige l'installation de MDAC (Microsoft Data Access Components). L'avantage de ce fournisseur est qu'il peut dialoguer avec n'importe quelle base de données le temps que le pilote OLE DB est installé ; Mais par rapport à SQL server, OLE DB utilise une couche logicielle nommée OLE DB. Il requiert donc plus de ressources diminuant par conséquent les performances. |
| ODBC        | Les classes de ce fournisseur se trouvent dans l'espace de nom <code>System.Data.Odbc</code> , chaque nom de ces classes est préfixé par <code>Odbc</code> . Tout comme l'OLE DB, ODBC exige l'installation de MDAC. Il fonctionne avec le même principe qu'OLE DB mais au lieu d'utiliser une couche logicielle, il utilise le pilote ODBC.  |
| Oracle      | Les classes de ce fournisseur se trouvent dans l'espace de nom <code>System.Data.OracleClient</code> , chaque nom de ces classes est préfixé par <code>Oracle</code> . Il permet simplement de se connecter à une source de données Oracle.   |

| Espace de nom                   | Fournisseur OLE DB                    |
|---------------------------------|---------------------------------------|
| <b>System.Data</b>              | Compatible avec tous les fournisseurs |
| <b>System.Data.OleDb</b>        | OLEDB                                 |
| <b>System.Data.SqlClient</b>    | SQL                                   |
| <b>System.Data.OracleClient</b> | Oracle                                |
| <b>System.Data.ODBC</b>         | ODBC                                  |

- Pour dialoguer avec la base de données, tous ces fournisseurs implémentent six classes de bases :

| Classe         | Description   |
|----------------|---|
| Command        | Stocke les informations sur la commande et permet son exécution sur le serveur de base de données.  |
| CommandBuilder | Permet de générer automatiquement des commandes ainsi que des paramètres pour un <i>DataAdapter</i> .   |
| Connection     | Permet d'établir une connexion à une source de données spécifiée.   |
| DataAdapter    | Permet le transfert de données de la base de données vers l'application et inversement (par exemple pour une mise à jour, suppression ou modification de données). Il est utilisé en mode déconnecté (voir partie 5.4). |
| DataReader     | Permet un accès en lecture seule à une source de données.   |
| Transaction    | Représente une transaction dans le serveur de la base de données.   |

# Mode connecté & mode déconnecté

- L'ADO.NET permet de séparer les actions d'accès ou de modification d'une base de données. En effet, il est possible de manipuler une base de données sans être connecté à celle-ci, il suffit juste de se connecter pendant un temps court afin de faire une mise à jour,
- Deux modes existent pour l'utilisation des données dans une application :
  - *Mode connecté : l'application client a un accès direct à la source de données*
  - *Mode déconnecté : il est possible de travailler sur des données sans avoir un accès direct et permanent à la base.*

| Mode       | Avantages   | Inconvénients  |
|------------|---|--|
| Connecté   | Avec un mode connecté, la connexion est permanente, par conséquence les données sont toujours à jour. De plus il est facile de voir quels sont les utilisateurs connectés et sur quoi ils travaillent. Enfin, la gestion est simple, il y a connexion au début de l'application puis déconnexion à la fin.                          | L'inconvénient se trouve surtout au niveau des ressources. En effet, tous les utilisateurs ont une connexion permanente avec le serveur. Même si l'utilisateur n'y fait rien la connexion gaspille beaucoup de ressource entraînant aussi des problèmes d'accès au réseau. |
| Déconnecté | L'avantage est qu'il est possible de brancher un nombre important d'utilisateurs sur le même serveur. En effet, ils se connectent le moins souvent et durant la plus courte durée possible. De plus, avec cet environnement déconnecté, l'application gagne en performance par la disponibilité des ressources pour les connexions. | Les données ne sont pas toujours à jour, ce qui peut aussi entraîner des conflits lors des mises à jour. Il faut aussi penser à prévoir du code pour savoir ce que va faire l'utilisateur en cas de conflits.  |

# Accès à une BD (SQL-Server)

- 1. Ouvrir une connexion**

---

- 2. Requêtes T-SQL (select, insert, update, ...)**
- 3. Fermer la connexion**

# Établir une connexion à une BD

- Il faut disposer d'un objet **Connexion** pour avoir accès à la base de données
- On met dans la propriété **ConnectionString** les paramètres de la base de données (nom de la base de données, chemin, mot de passe..).
- En fonction de la BD les paramètres sont différents.
- Avec la méthode **Open** on ouvre la base.
- Avec la méthode **Close** on ferme la base.

| Fournisseur OLE DB | Classe          |
|--------------------|-----------------|
| ODBC               | OdbcConnection  |
| OLEDB              | OleDbConnection |
| SQL Server         | SqlConnection   |

Parmi les propriétés de ces classes

| Propriétés       | Description   |
|------------------|---|
| ConnectionString | Chaîne utilisée pour la connexion contenant le fournisseur OLE DB et le chemin d'accès de la BD |
| State            | Etat de la connexion  |

|                       |  |
|-----------------------|--|
| Data Source           | Indique le nom ou l'adresse réseau du serveur.   |
| Initial Catalog       | Indique le nom de la base de données où l'application doit se connecter.                                       |
| Integrated Security   | Indique s'il faut un nom et un mot de passe. Si la valeur est sur False, un login et password seront demandés. |
| Max Pool Size         | Indique le nombre maximum de connexion dans un pool. Par défaut, le nombre maximum de connexions est 100.      |
| Min Pool Size         | Indique le nombre minimum de connexion dans un pool.   |
| Persist Security Info | Indique si le nom et le mot de passe est visible par la connexion.   |
| Pwd                   | Indique le mot de passe associé au compte SQL Server.  |
| Pooling               | Indique si une connexion peut être sortie d'un pool.   |
| User ID               | Indique le nom du compte SQL Server.   |

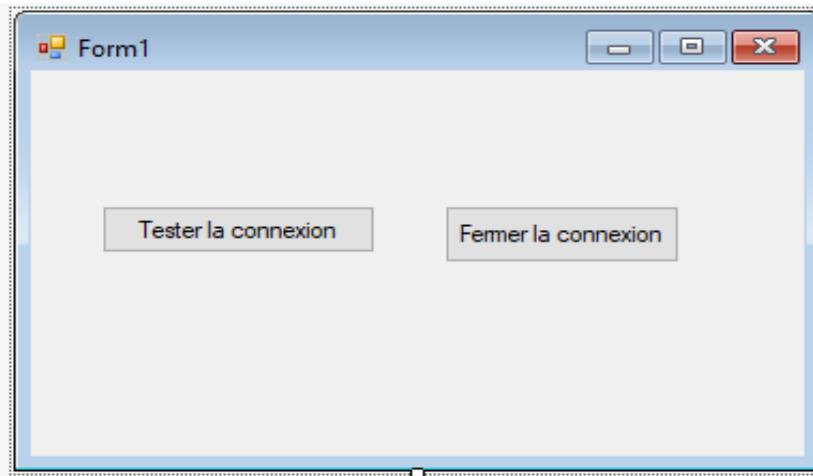
| Propriété  | Description   |
|------------|---|
| Broken     | Permet de savoir si la connexion est interrompue, cette connexion peut se fermer puis se rouvrir. |
| Closed     | Permet de savoir si l'objet connexion est fermé.  |
| Connecting | Permet de savoir si l'objet connexion est en cours de connexion.                                  |
| Executing  | Permet de savoir si une commande est en train de s'exécuter.                                      |
| Fetching   | Permet de savoir si l'objet connexion est en train de récupérer des données.                      |
| Open       | Permet de savoir si l'objet connexion est ouvert.   |

# Exemple

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace TestConnexionSQLServer
{
    public partial class Form1 : Form
    {
        // 'Create ADO.NET objects.
        SqlConnection myConn;

        public Form1()
        {
            InitializeComponent();
        }
    }
}
```



```
private void button1_Click(object sender, EventArgs e)
{
    myConn = new SqlConnection("Data Source=DESKTOP-P59KUOO;
Initial Catalog = COMMANDE;Integrated Security=True");
    myConn.Open();
    if (myConn.State == ConnectionState.Open)
        MessageBox.Show("Bien connecté!!!");
    else
        MessageBox.Show("Problème de connexion");
}

private void button2_Click(object sender, EventArgs e)
{
    myConn.Close();
    if (myConn.State == ConnectionState.Closed)
        MessageBox.Show("Connexion bien fermée!!!");
    else
        MessageBox.Show("Problème de fermeture de connexion");
}
```

# Objet COMMAND

- Les commandes contiennent toutes les informations nécessaires à leur exécution et effectuent des opérations telles que créer, modifier ou encore supprimer des données d'une base de données. Vous utilisez ainsi des commandes pour faire des exécutions de requêtes T-SQL qui renvoient les données nécessaires.

Chaque fournisseur de base de données possède leurs propres objets *Command* qui sont les suivantes :

| Nom           | Type de sources de données |
|---------------|----------------------------|
| SqlCommand    | SQL Server                 |
| OleDbCommand  | OLE DB                     |
| OdbcCommand   | ODBC                       |
| OracleCommand | Orade                      |

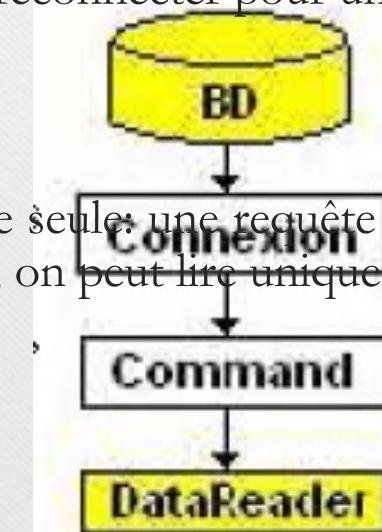
Il existe plusieurs propriétés et méthodes communes à chaque fournisseur pour gérer des commandes, voici les principales :

| <u>Propriétés</u> |   |
|-------------------|---|
| Nom               | Description   |
| CommandText       | Permet de définir l'instruction de requêtes SQL ou de procédures stockées à exécuter.<br>Lié à la propriété <i> CommandType</i> .   |
| CommandTimeout    | Permet d'indiquer le temps en secondes avant de mettre fin à l'exécution de la commande.  |
| CommandType       | Permet d'indiquer ou de spécifier la manière dont la propriété <i> CommandText </i> doit être exécutée.   |
| Connection        | Permet d'établir une connexion.   |
| Parameters        | C'est la collection des paramètres de commandes. Lors de l'exécution de requêtes paramétrées ou de procédures stockées, vous devez ajouter les paramètres objet dans la collection. |
| Transaction       | Permet de définir la <i> SqlTransaction </i> dans laquelle la <i> SqlCommand </i> s'exécute.  |

| <u>Méthodes</u>  |   |
|------------------|---|
| Nom              | Description   |
| Cancel           | Permet de tenter l'annulation de l'exécution d'une commande.                                      |
| ExecuteNonQuery  | Permet d'exécuter des requêtes ou des procédures stockées qui ne retournent pas de valeurs.       |
| ExecuteReader    | Permet d'exécuter des commandes et les retourne sous forme de tableau de données (ou des lignes). |
| ExecuteScalar    | Permet d'exécuter les requêtes ou les procédures stockées en retournant une valeur unique.        |
| ExecuteXMLReader | Permet de retourner les données sous le format XML.   |

# Mode connecté

- Le **DataReader** est un objet ADO.NET qui permet de lire très rapidement une table, enregistrement par enregistrement, du début à la fin. Il n'y a pas possibilité d'écrire dans la base
- Le **DataReader** est en lecture seule, les données lues dans la BD sont accessibles dans le **DataReader** seul, c'est-à-dire qu'il est impossible de revenir en arrière sur les enregistrements lus. Il n'a été créé que pour la lecture pure et simple de données.
- Le **DataReader** doit toujours être associé à une connexion active, c'est-à-dire qu'il ne peut pas se déconnecter, effectuer quelque chose puis se reconnecter pour une mise à jour.
- Avec un objet **DataReader** on extrait les données en lecture seule: une requête SQL (sur un objet *command*) charge le **DataReader**. C'est rapide; on peut lire uniquement les données et aller à l'enregistrement suivant.



# **DataReader (SqlServer)**

---

- Dans ce cas, il faut suivre les étapes suivantes :
  - Créer un objet **SqlConnection**
  - Ouvrir la connexion
  - Créer un objet **SqlCommand**
  - Créer l'objet **DataReader** en exécutant la méthode **ExecuteReader**
  - Parcourir les enregistrements retournés par la méthode **read**.

## les méthodes de cet objet

| Méthode  | Description   |
|--|---|
| Close  | Ferme le DataReader   |
| Read   | Avance au prochain enregistrement, retourne True s'il existe d'autres enregistrements et False s'il n'y en a plus |
| GetValue(i)  | Retourne la valeur du champ sans type à l'indice « i »  |
| GetString, GetDouble,<br>GetBoolean, GetInt32, ... | Retourne la valeur du champ typé à l'indice « i »   |
| GetType(i)   | Retourne le type du champ à l'indice « i »  |
| GetName(i)   | Retourne le nom du champ à l'indice « i »   |
| GetOrdinal(nom)                                    | Retourne l'indice du champ donné en paramètre   |
| FieldCount   | Retourne le nombre de colonnes du DataReader  |

# Exemple

Form2

Remplir

|   | nom produit | libellé | prix produ |
|---|-------------|---------|------------|
| ▶ | 1           | P1      | 150        |
|   | 2           | P2      | 100        |
|   | 3           | P3      | 120        |
|   | 4           | P4      | 100        |
|   | 5           | P5      | 110        |

Nombre produits 14

Autre

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
namespace TestConnexionSQLServer
```

```
{
```

```
    public partial class Form2 : Form
    {
        SqlDataReader myReader;
        String results;
```

```
        SqlConnection myConn = new SqlConnection("Data
Source=DESKTOP-P59KUOO;Initial Catalog =
COMMANDER;Integrated Security=True");
```

```
        SqlCommand myCmd;
```

```
        public Form2()
        {
            InitializeComponent();
        }
    }
```

```
    private void button1_Click(object sender, EventArgs e)
```

```

    {
        myConn.Open();
        myCmd = myConn.CreateCommand();
        myCmd.CommandText = "SELECT liblle FROM
COMMANDER.dbo.Produit";
        SqlDataReader myReader =
myCmd.ExecuteReader();
        // remplir la listebox par le résultat de la requête
        while (myReader.Read())
            Liste.Items.Add(myReader.GetString(0));
        myReader.Close();
        myConn.Close();
    }
}
```

```
//remplir le datagridview en mode connecté (par datareader)
    myConn.Open();
    SqlCommand MyCommand = myConn.CreateCommand();
    MyCommand.CommandText = "SELECT num_prod, liblle, pu_prod FROM COMMANDE.dbo.Produit";
    SqlDataReader myRd = MyCommand.ExecuteReader();

    // On récupère les informations sur le schéma de la colonne dans la base
    dataGridView1.DataSource = null;
    dataGridView1.Columns.Clear();
    dataGridView1.ColumnCount = 3;
    dataGridView1.Columns[0].Name = "nom produit";
    dataGridView1.Columns[1].Name = "libellé";
    dataGridView1.Columns[2].Name = "prix produit";

    while (myRd.Read())
        dataGridView1.Rows.Add(myRd[0], myRd[1], myRd[2]);
    myRd.Close();

    SqlCommand cmdcount = myConn.CreateCommand();
    cmdcount.CommandText = "SELECT count(*) FROM COMMANDE.dbo.Produit";
    label2.Text = cmdcount.ExecuteScalar().ToString();
    myConn.Close();

}
```

- Ajout d'un nouveau enregistrement

```
private void button3_Click(object sender, EventArgs e)
{
    myCmd = myConn.CreateCommand();
    myCmd.CommandText = " insert into COMMANDE.dbo.Produit values
('P22', 150, 5, 6, 15)";
    myConn.Open();
    myCmd.ExecuteNonQuery();
    myConn.Close();
}
```

# Exercice

The screenshot shows a Windows application window titled "Form3". Inside the window, there are four text input fields arranged vertically. Each field has a label to its left and contains a value:

- label: numéro produit, value: 1
- label: nom produit, value: P1
- label: prix unitaire, value: 150
- label: quantité stock, value: 20

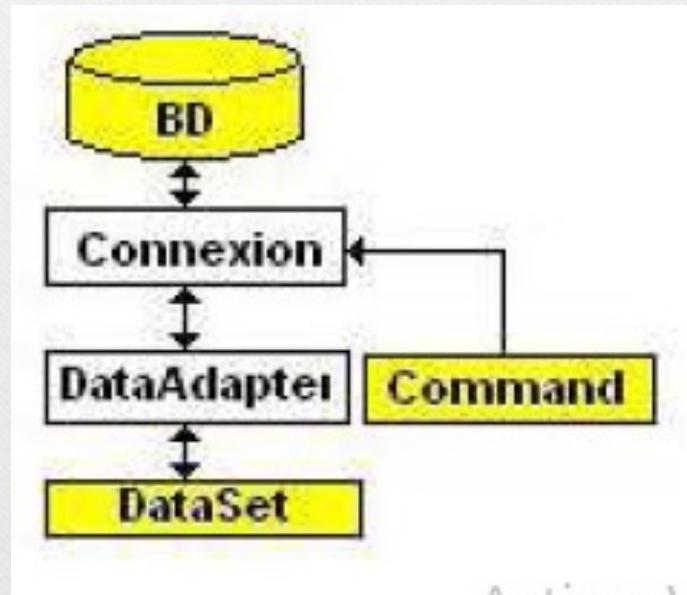
Below these fields are two buttons:

- previous
- next

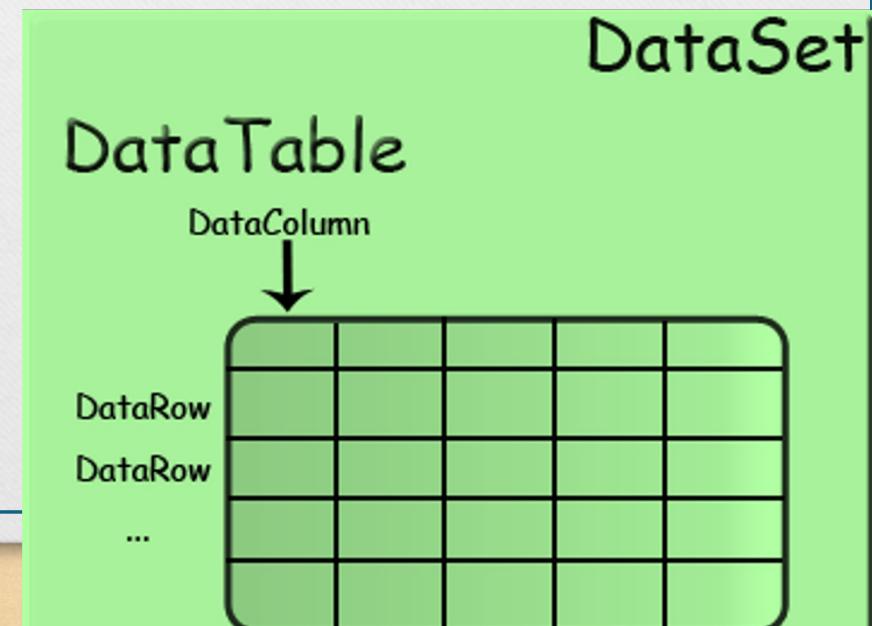
- Écrire le code de l'interface suivante qui permet de parcourir la table produit en utilisant les boutons ‘précédent’ et ‘suivant’.

# Mode déconnecté

- Le travail en mode déconnecté signifie que la base de données est copiée localement; donc la connexion est libérée après chargement du **DataSet**.
- Le **DataSet** en mode déconnecté: on ouvre la connexion, on le charge puis on ferme la connexion (il faut le faire, ce n'est pas automatique), on travaille sur le DataSet, on peut le réouvrir plus tard pour les mises à jour.



- Le **DataSet** est stocké dans l'espace de nom *System.Data*. C'est un cache de données en mémoire, c'est-à-dire qu'il permet de stocker temporairement des données utilisées dans votre application.
- Le **DataSet** contient la collection d'objets **DataTable** qui peuvent être liés avec les objets **DataRelation**.
- Dans le cas du mode déconnecté, cet objet va nous permettre d'importer la partie désirée de la base de données (fonction de la requête de sélection) en local.
- Ainsi grâce à des objets nécessaires à la connexion classique (*commande select, connections string...*) et un **DataAdapter**, nous pourrons relier ("Binder") un *DataSet* sur une base de données (en lecture et en écriture grâce à une méthode de mise à jour de la base de données(*update*)).



### les éléments d'un *DataSet* :

| Objet      | Description  |
|------------|--|
| DataTable  | Correspond à une table. Contient une collection de DataColumn et de DataRow. |
| DataColumn | Représente une colonne de la table.  |
| DataRow    | Correspond à un enregistrement de la table.                                  |

Form1

## LISTE DES LIVRES

|   | RefLiv  | Titre                | Domaine         | Editeur        | Nbpage | Annee |
|---|---------|----------------------|-----------------|----------------|--------|-------|
| ▶ | A04540  | Système d'informa... | Base de données | Taylor&Francis | 102    | 2001  |
|   | A1020   | Statistique          | MathApplique    | Taylor&Francis | 202    | 2001  |
|   | A54152  | Statistique          | MathApplique    | Sciences       | 54     | 2010  |
|   | AB0054a | Lavie                | Roman           | Folio          | 82     | 2012  |
|   | An546   | Analyse de Donn...   | MathApplique    | DataSciences   | 52     | 2010  |

Ajouter      Supprimer      Mettre à jour      Chercher

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace ParcoursDataReader
{
    public partial class Form1 : Form
    {
        SqlConnection sqlcon = new SqlConnection(" Data Source=DESKTOP-P59KUOO;Initial Catalog=Bibliotheque_2020;Integrated Security=True");
        SqlCommand sqlcmd;
        SqlDataAdapter sqldta = new SqlDataAdapter();
        DataSet dst = new DataSet();
        String str;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            str = "select * from Bibliotheque_2020.dbo.Livre";
            sqlcmd = sqlcon.CreateCommand();
            sqlcmd.CommandText = str;
            sqldta.SelectCommand = sqlcmd;
            sqldta.Fill(dst, "ListeLivres");
            dataGridView1.DataSource = dst.Tables["ListeLivres"];
        }
    }
}
```

# Exercice

- Nouveau enreg: (x025, ‘ProgrammationC’, ‘Programmation’, ‘Taylor’, 50, 2010)
  - ➔ String str= “Insert into COMMANDE.dbo.Produit(x025, ‘ProgrammationC’, ‘Programmation’, ‘Taylor’, 50, 2010) ”;
  - ➔ Sqlcom.ExecuteNonQuery();
- Compléter l’interface précédente pour l’ajout, la suppression, la mise à jour et la recherche d’un enregistrement