# ADO .NET

# SOMMAIRE

# AVANT PROPOS

ADO.NET propose un accès cohérent à des sources de données, telles que SQL Server et XML, ainsi qu'à des sources de données exposées via OLE DB et ODBC. Des applications grand public de partage de données peuvent utiliser ADO.NET pour se connecter à des sources de données et extraire, manipuler et mettre à jour les données qu'elles contiennent.

ADO.NET sépare l'accès aux données de leur manipulation en composants distincts qui peuvent être utilisés individuellement ou en tandem. ADO.NET comprend des fournisseurs de données .NET Framework pour la connexion à une base de données, l'exécution de commandes et l'extraction de résultats. Ces résultats sont traités directement, placés dans un objet DataSet ADO.NET pour pouvoir être exposés à l'utilisateur de manière adéquate, combinés aux données de différentes sources ou passées entre couches. L'objet **DataSet** peut également être utilisé indépendamment d'un fournisseur de données .NET Framework pour gérer des données locales pour l'application ou provenant de XML.

Le but de ce cours n'est pas d'étudier en profondeur tous ces éléments, mais de fournir les éléments et techniques permettant d'écrire des applications sous .net et plus précisément permettant l'accès aux données sous ADO.net. Nous utiliserons le langage C# pour illustrer nos exemples.

Par conséquent, à la fin de ce cours, l'étudiant devra être capable de :

- D'identifier les composants de la couche ADO.NET ainsi que le rôle
- D'identifier les composants nécessaires à la connexion à une BD spécifique
- D'écrire des codes permettant l'accès aux sources de données (Relationnel, XML)
- D'interfacer une application graphique à une source de données .NET

# CHAPITRE I VUE D'ENSEMBLE D'ADO.NET

## I *Présentation de ADO.NET*

ADO.net est une évolution logique des techniques d'accès aux données dont les principales sont:

- ODBC(Open Database Connectivity)
- DAO(Data Acces Object)
- RDO(Remote Data Object)
- ADO(Activex Data Object)

Il est constitué d'un ensemble de classe positionné dans le namespace(équivalent d'un package en C) System.Data. Il est par conséquent utilisé par tous les langages de ce framework.

ADO.NET propose donc un modèle qui intègre les fonctionnalités de XML et de ADO C'est-à-dire qu'il offre des ensembles de classes, structures, interfaces, etc faisant partie intégrante du framework .NET et permettant un accès efficace et cohérent aux données. Les avantages apportés par ADO.NET sont essentiellement:

- La montée en charge pose moins de problèmes qu'auparavant. Par exemple il est vraiment aisé de dédoubler un serveur web en cas de puissance de traitement accrue. Ceci est en partie dû à la possibilité de travailler en mode déconnecté avec ADO.NET et d'utiliser des caches en mémoire.
- L'utilisation abondante de XML (eXtensible Markup Language), une technologie portable et standardisée permettant des échanges de données indépendants des systèmes et des plateformes.
- Les facilités offertes au niveau du développement. Le Framework .NET, et donc les langages supportés, ont été pensés et conçus pour intégrer la gestion des bases de données. Visual Studio intègre tout ce dont le développeur a besoin, quels que soient les types d'applications.

L'accès aux données sous le Framework .NET se présente comme suit :

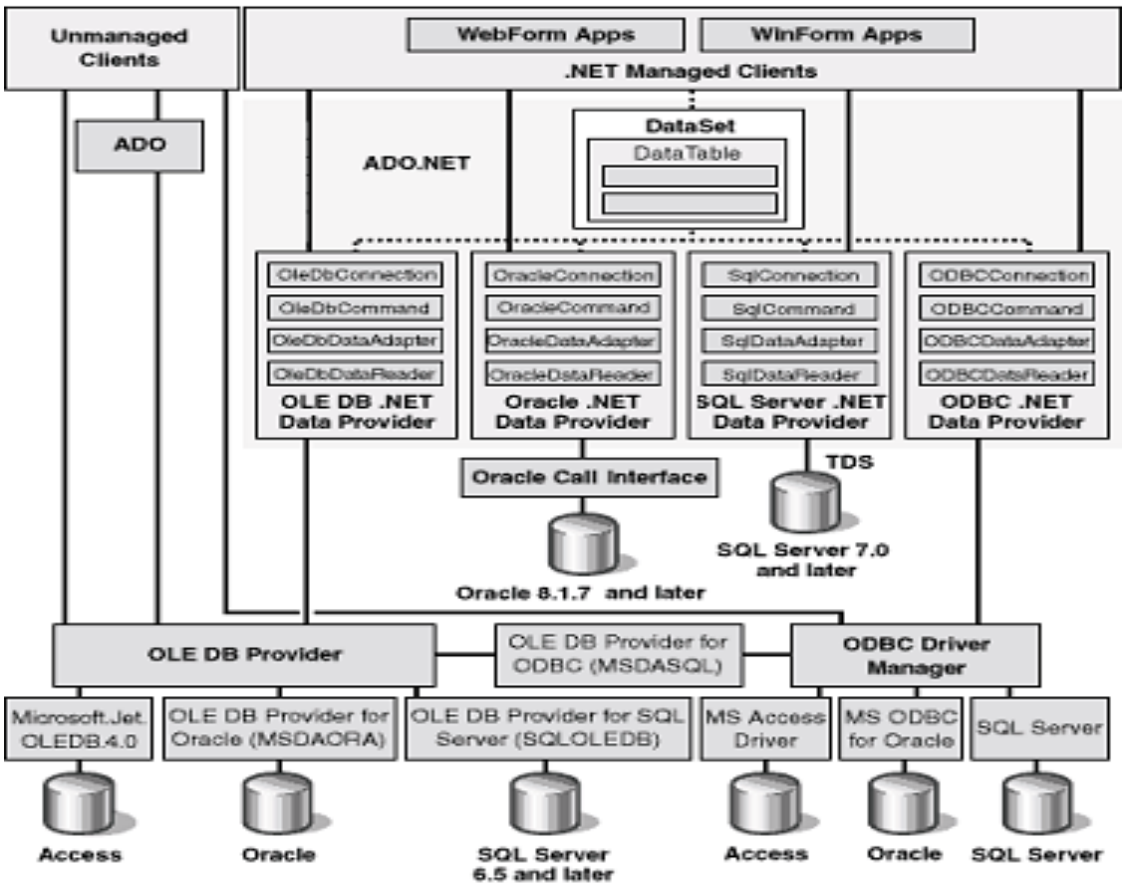


**Figure 1:Vue d'ensemble d'accès aux données sous .NET**

Si on s'intéresse au seul Framework ADO.NET, alors ce schéma se résume en:

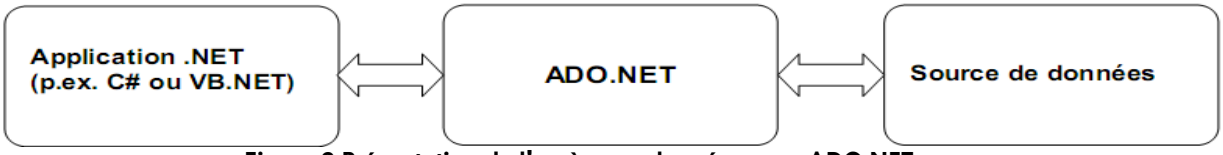


**Figure 2:Présentation de l'accès aux données sous ADO.NET**

Il est à noter qu'il est toujours possible d'utiliser les objets ***ADO (ADODB….)*** dans une application .net

## II Mode d'accès à une base de données

La vue d'ensemble d'accès à une source de données sous ADO.NET est la suivante :

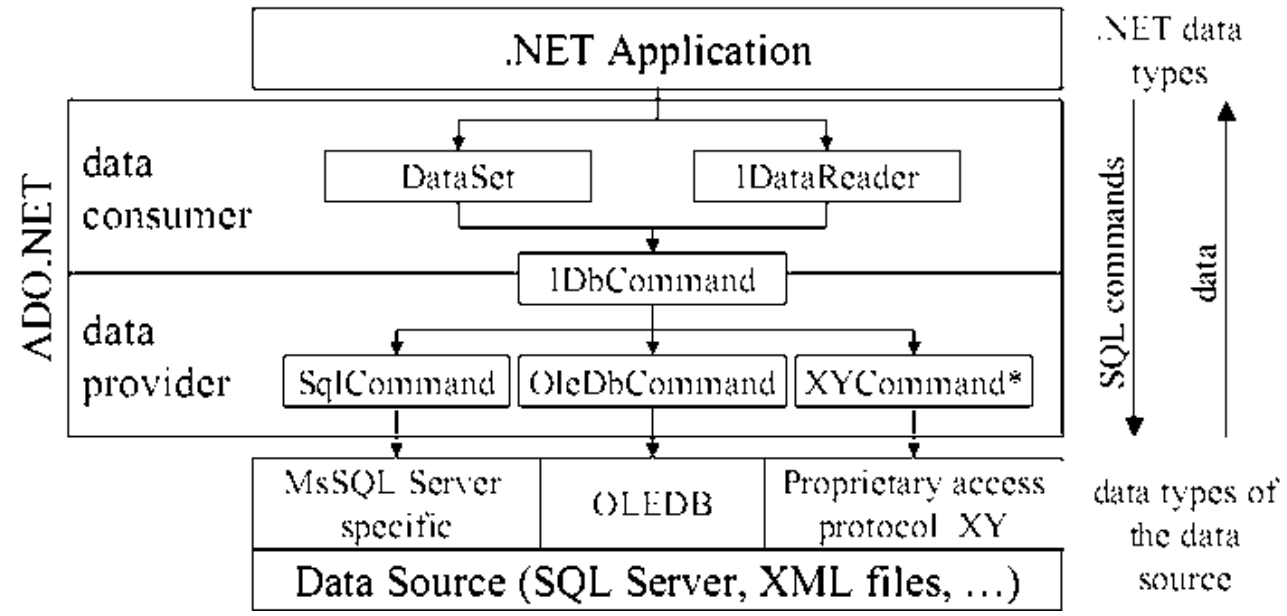


**Figure 3: Schema de l'accès aux données sous ADO.NET**

L'interaction entre une application et une source de données sous ADO.NET, peut se faire sous deux modes :

- ***Le mode connecté***: Dans ce mode, une application ouvre une connexion avec une base de données et reste connectée pendant le traitement des données. Il s'agit du mode le plus répandu et habituel. Dans ce mode, une application (client) "interagit" avec la base de données (serveur), demande des données, les modifie et les met à jour au fur et à mesure, avance enregistrement par enregistrement dans une table, etc. Il y a un lien fort, au sens interactivité du terme, entre le client et le serveur.
- ***Mode déconnecté:*** Dans le mode déconnecté, une application se connecte à une base de données, rapatrie une partie des données et se déconnecte de la base de données. S'il s'agit d'une opération de lecture seule (comme par exemple lors d'une consultation sur un site internet) l'opération est terminée. Si toutefois l'application doit effectuer des adjonctions ou mises à jour de données, elle doit se reconnecter à la base pour cela. Ce mode est particulièrement bien adapté à des applications internet dans lesquelles, par essence, on ne peut pas imposer une connexion continue à un client.

  Une application peut faire usage des deux modes de connections. Les avantages et inconvénients de ces différents modes sont les suivants :

| | ***Mode connecté*** | ***Mode déconnecté*** |
|---|---|---|

| | | |
|---|---|---|
| **Avantages** | • *Accès concurrent plus facile à contrôler*<br>• *Données plus pertinentes (globalement plus souvent à jour)* | • *Bien adapté aux utilisateurs mobiles*<br>• *Augmentation des performances et facilité de montée en charge*<br>• *Utile dans les cas de consultation de Données* |
| **Inconvénients** | • *Nombre d'accès à la base de données et trafic réseau plus élevés* | • *Les données sont moins souvent à jour*<br>• *Nécessité de résoudre les conflits lors des mises à jour*<br>• *Demande de la place mémoire côté client et plus de subtilité afin d'optimiser la quantité d'information rapatriée sur le client* |

## III Les objets d'ADO.NET

Comme nous l'avons présenté précédemment, les objets d'ADO.Net sont divisés en deux catégories. Un objet particulier fait la liaison entre ces deux catégories (l'objet DataAdapter) comme le présente la figure ci-dessous. Il est à noter qu'une application peut fonctionner avec les deux modes de connexion.

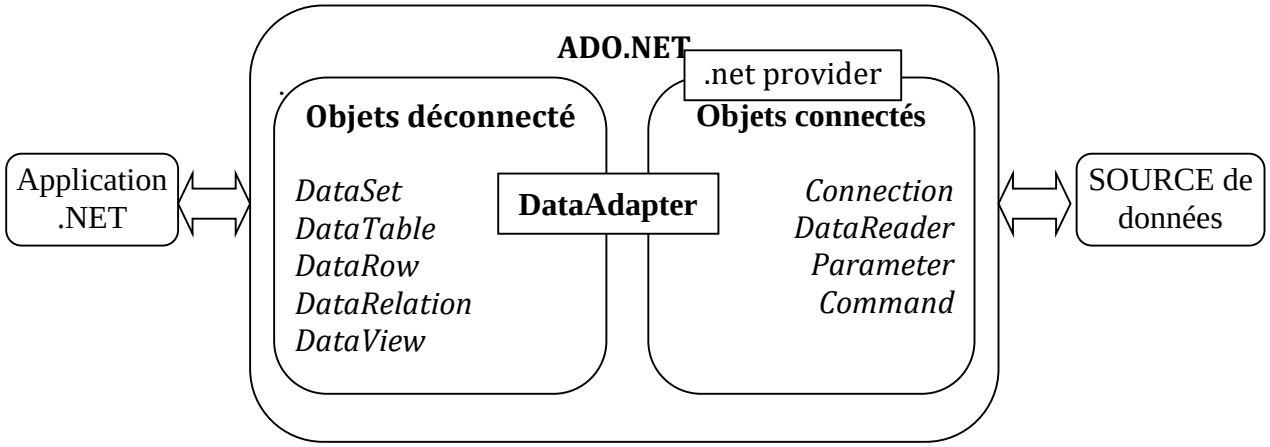


**Figure 4: Les objets sous ADO.Net**

# CHAPITRE II TRAVAILLER EN MODE CONNECTE

## I Introduction

Le schéma classique de travail en mode connecté est le suivant :

- *Ouvrir la connexion à une BD*
- *Exécuter des requêtes sur la BD*
- *Fermer la connexion*

Pour effectuer ces différents traitements, il faut utiliser un fournisseur d'accès (data provider), qui est spécifique à chaque BD. Il est possible d'écrire un fournisseur d'accès pour chaque BD, toutefois, la plateforme .NET est fourni en standard avec quelques fournisseurs spécifiques:

- Le fournisseur **SQL**Client de l'espace de nom System.Data.SqlClient qui est le fournisseur par référence d'une BD SQL SERVER.
- Le fournisseur **ORACLE**Client de l'espace de nom System.Data.ORACLEClient qui est le fournisseur par référence d'une ORACLE.
- Le fournisseur **ODBC** de l'espace de nom System.Data.Odbc qui est permet la connexion à une BD via le pont ODBC.
- Le fournisseur **Oledb** de l'espace de nom System.Data.Oledb qui permet la connexion à une BD en transitant via Oledb.

Il existe sur le marché des fournisseurs d'accès spécifiques à certains SGBD. C'est le cas de MYSQL qui propose un fournisseur dont les classes sont positionnées dans le namespace ***MySql.Data.MySqlClient***

Chaque fournisseur d'accès comporte les classes ci-dessous : ***Command, Connection, Parameter, DataAdapter, DataReader, Transaction.*** Ces classes dépendent du fournisseur d'accès aux données. A titre d'exemple, on aura ***SqlConnection, SqlCommand, OledbConnection, OledbCommand,….***

## II L'objet Connection

Cet objet permet d'établir une connexion avec une source de données. Cette connexion nécessite un fournisseur d'accès (data provider) que nous nommons **XY** et une chaine de connexion qui dépend du fournisseur d'accès. Les principales propriétés de l'objet **Connection** sont:

| **Propriétés ou méthodes** | **Explication** |
|---|---|
| ConnectionString | Définit la chaine de connexion à la BD. Joue le rôle d'URL |
| State | Indique l'état courant de la connexion. Les valeurs possibles pour cet objet sont :<br>• ***Closed*** : Base de données fermée<br>• ***Connecting*** : Lorsque la méthode open est appelé et avant qu'elle se ferme<br>• ***Open*** :Lorsque l'ouverture s'est terminée avec succès<br>• ***Executing*** : Une commande est en cours d'exécution sur l'objet Connection<br>• ***Fetching*** :Les données sont en cours d'extraction |
| Open() | Permet d'ouvrir la connexion |
| Close() | Ferme la connexion à la BD |

En résumé, pour effectuer la connexion à la BD, on utilise l'un des bouts de code ci-dessous ::

```
String chaine_connexion="…..";
XYConnection con =new XYConnection( )
con.connectionString=chaine_de_connexion
con.open( )
```

```
String chaine_connexion="…..";
XYConnection con = new XYConnection(chaine_connexion)
con.open( )
```

Le tableau ci-dessous présente le schéma général de la chaîne de connexion pour quelques fournisseurs :

| Fournisseur | Structure de la chaîne de connexion |
|---|---|
| SQLclient | *"**Initial Catalog**=nomSchemaBD;**Data Source**=adresse_serveur;**user id**=compte;**password**=mot_de_passe"* |
| Oledb (pour SQL Server) | *"**provider**=**sqloledb**;**Initial Catalog**=nomSchemaBD;**Data Source**=adresse_serveur;**user id**=compte; **password**=mot_de_passe"* |
| Oledb(Pour Access 2003) | *"**Provider**=**Microsoft.Jet.OLEDB.4.0**;**Data Source**=nom_complet_BD; "* |
| MySql | *"**Database**=nom_BD;**Data Source**=nom_serveur;**User Id**=compte;**Password**=mot_passe";*<br>*Pour ce SGBD, on dispose d'autres attributs comme: **port, Convert Zero Datetime, Allow Zero Datetime*** |

On obtient ainsi les bouts de code ci-dessous :

- Pour une BD SQL SERVER à travers le provider SQLclient:

  *String chaine_connexion = "Initial Catalog=Northwind;Data Source=localhost;user id=sa password='' "*
  *System.Data.SQLClient.SqlConnection con=new System.Data.SQLClient.SqlConnection(chaine_connexion)*

- Pour une BD SQL SERVER à travers le provider Oledb:

  *String chaine_connexion = "provider=sqloledb; Initial Catalog=Northwind;Data Source=localhost;user id=sa password='' "*
  *System.Data.Oledb.OledbConnection con= new System.Data.Oledb.OledbConnection(chaine_connexion)*

- *Pour une BD Access*

  *OleDbConnection con = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=bdTest.mdb; ");*

- *Pour une BD MySql*:

  *String chaine_connexion = "Database=Test;Data Source=localhost;User Id=username;Password=pass";*
  *MySqlConnection con = new MySqlConnection(chaine_connexion);*

## III L'objet Command

Une fois la connexion établit, on peut exécuter des requêtes. L'exécution des requêtes se fait via l'objet Command, qui s'appuie sur un objet connexion. Cette classe hérite de l'interface IDbCommand. La commande à exécuter peut être une requête de sélection, de mise à jour, DDL ou LCD, une procédure ou une fonction stockée.

Les principales propriétés et méthodes d'un objet Command sont:

| Propriétés ou méthodes | Explication |
|---|---|
| *CommandText* | Définit la commande à exécuter : requête de sélection, mise à jour, procédure stockée, … |
| *Transaction* | Définit l'objet transaction à utiliser |
| *Connection* | Indique la connexion utilisée pour exécuter la commande |
| ***executeScalar( )*:** | exécuter une commande renvoyant une seule valeur |
| ***executeReader( )*** | exécuter une commande renvoyant un jeu d'enregistrement (Typiquement les requêtes Select). Le résultat est un objet DataReader |
| ***executeNonQuery()*** | d'exécuter les autres types de commandes (ordre DDL, requêtes de mise à jour, …) |

Avant l'exécution d'une commande, les propriétés ***CommandText*** et ***Connection*** doivent être définies. Pour créer un objet command, on peut utiliser l'un des codes ci-dessous :

```
XYCommand cmd;
        cmd =new XYCommand(requite_a_executer,connexion_a_utiliser)

XYCommand cmd = con.createCommand()
```

### III.1 La méthode executeScalar()

Cette méthode est nécessaire pour exécuter des commandes renvoyant une seule information. Cette méthode renvoie une information de type Object qu'il faut convertir.

**Exemple :**

Soit la table Employe(**mle**, nom, prenom, sexe, salBase), de la BD bdTest.mdb écrire un code qui renvoie le plus grand salaire de base.

```
OleDbConnection con = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=bdTest.mdb; ");
OleDbCommand cmd=new OleDbCommand("Select max(salBase) from employe",con);
decimal maxSalaire=decimal.Parse(cmd.executeScalar().ToString());
Console.WriteLine(maxSalaire);
```

*Pour lire des BLOB à partir d'une commande, on utilise:*

```
Byte[ ] blob=(byte[ ])cmd.exeeceuteScalar();
System.IO.MemoryStream str=new System.IO.MemoryStream();
str.write(blob,0,blob.Length);
Bitmap bmp=new Bitmap(str);
```

### III.2 La méthode executeNonQuerry( )

Elle est utilisée pour exécuter des commandes qui ne renvoient aucune valeur (CREATE TABLE, ALTER TABLE, CREATE DATABASE). Cette fonction permet dans le cas des requêtes de mises à jour (INSERT, UPDATE, DELETE) d'indiquer le nombre d'enregistrements affectés.

**Exemple :** En utilisant le schéma de la table employé précédente, écrire un code qui double le salaire des employés dont le salaire est inférieur à 80 000.

```
OleDbConnection con = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=bdTest.mdb; ");
OleDbConnection cmd=new OleDbConnection("UPDATE EMPLOYE SET salbase=2*salBase where salbase<80000",con);
int n=cmd.executeNonQuerry();
Console.WriteLine(n+"Lignes de modifies");
```

### III.3 La méthode executeReader( )

Elle est utilisée pour exécuter des commandes renvoyant des jeux d'enregistrement, typiquement des requêtes de sélection. Le résultat e ce type de requête est un objet de type DataReader ( voir IV)

## IV L'objet DataReader

Un DataReader est un jeu de donnée en mode connecté. Il est beaucoup plus rapide qu'un Recordset, est accessible uniquement en lecture et permet un parcours uniquement vers l'avant. Elle dispose des propriétés et méthodes:

| Propriétés ou méthodes | Explication |
|---|---|
| ***FieldCount*** | renvoie le nombre de colonne |
| ***IsClosed*** | Indique si l'objet DataReader est ouvert |
| ***read()*** | Pour se déplacer vers l'enregistrement suivant. Elle renvoie true s'il existe un prochain enregistrement et faux dans le cas contraire. Il est à noter qu'un DataReader charge une ligne à la fois. |
| ***GetName(int i)*** | Obtient le nom de la colonne à l'indice i |
| ***GetSchemaTable()*** | Pour le descriptif du schema |
| ***GetXX(int indice) ou GetXX(String colName)*** | où XX représente le type de la donnée. Elle permet de récupérer la valeur de la colonne de rang i( à partir du rang 0) ou dont le nom est colName. |

| | |
|---|---|
| ***isDbNull(int i)*** | Indique si la colonne de rang i de la ligne active contient la valeur NULL |
| ***Close( )*** | Pour fermer le DataReader |

***Remarque :***

- On a au plus un DataReader ouvert par objet Connection.
- Si dr est un DataReader, dr[int i] est équivalent à dr.getObject(int i). dr[String col] est équivalent à dr.getObject(String col).
- Pour parcourir les lignes d'un DataReader de nom dr, on utilise le code :

```
while (dr.Read() ){
        //Traiter la ligne par dr[…] ou dr.getXX(…)
}
```

*Exemple: Ecrire un code qui affiche le matricule, nom et sexe des employés.*

```
OleDbConnection cn =new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=test.mdb");
cn.Open();
OleDbCommand cmd  = new OleDbCommand("SELECT * FROM Employees", cn);
OleDbDataReader dr = cmd.ExecuteReader();
while (dr.Read()) {
        Console.WriteLine (dr["mle"] + " " + dr["nom"]+"  "+dr.GetString("sexe")););
}
dr.Close();
cn.close();
```

## V L'objet Parameter

Afin d'augmenter la flexibilité d'une commande, il est prévu qu'elles puissent être paramétrées. L'objet ***Parameter*** permet justement de spécifier ces paramètres. Selon le SGBD, un paramètre dans la requête est précédé de @ ou de ?.

Pour ajouter un paramètre à une requête, on utilise la méthode ***add*** *(que l'on peut obtenir à partir de la propriété* ***Parameters*** d'un objet *Command)* la collection de paramètre, la propriété Value permet de définir la valeur du paramètre.

***Exemple :*** Ecrire une méthode qui prend en entrée le matricule, nom, prénom, sexe et salaire de base d'un employé, et inserre l'employé dans la BD.

```
Public void inserrerEmploye(String mle, String nom, String prenom, String sexe, double salaireBase){
        OleDbCommand cmd = new OleDbCommand("INSERT INTO Employees VALUES(@mle, @nom,@prenom,@sexe, @salaire)", con);
        cmd.Parameters.Add("@mle", OledbType.VarChar, 10);
        cmd.Parameters("@mle").Value=mle
        cmd.Parameters.Add("@nom", OledbType.VarChar, 50);
        cmd.Parameters("@nom").Value=nom
        cmd.Parameters.Add("@prenom", OledbType.VarChar, 50);
        cmd.Parameters("@prenom").Value=prenom
        cmd.Parameters.Add("@sexe", OledbType.Char, 1);
        cmd.Parameters("@sexe").Value=prenom
        cmd.Parameters.Add("@salaire", OledbType.Double);
        cmd.Parameters("@salaire").Value=salaireBase
        cmd.executeNonQuerry();
}
```

# CHAPITRE III Les objets du mode déconnecté

## I Introduction

Les objets du mode déconnecté permettent une liaison en mode déconnecté à une source de données. Ils sont indifférents du fournisseur d'accès. Les différents objets de ce mode peuvent se résumer en :

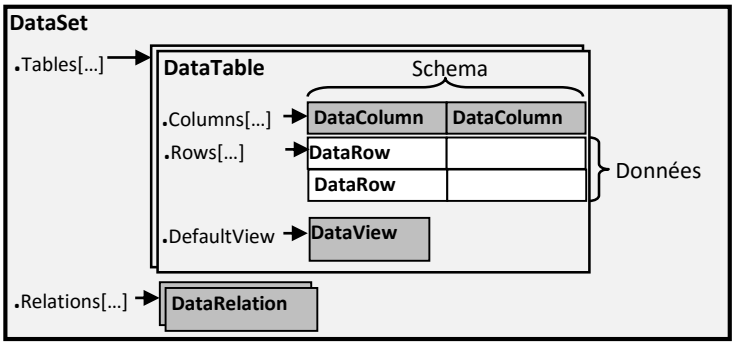


**Figure 5: Objets du monde déconecté.**

La logique de travail par contre est la suivante :

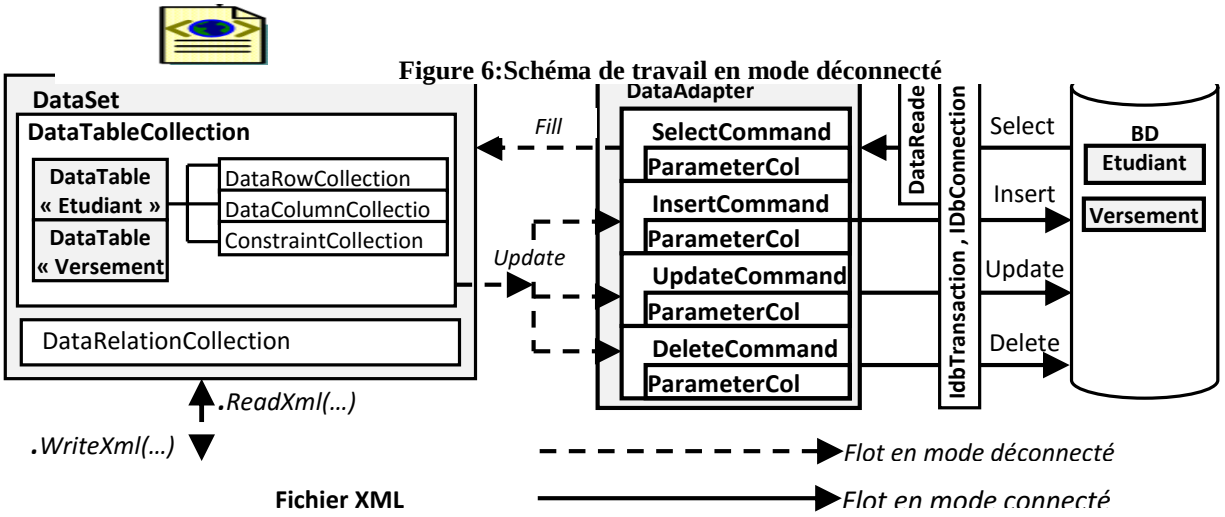


**Figure 6:Schéma de travail en mode déconnecté**

## II L'objet DataColumn

L'objet DataColumn représente une colonne en mode déconnecté. C'est un élément de la collection des colonnes d'un DataTable. Les principales propriétés de cet objet sont inscrites dans le tableau ci-dessous

| Propriétés | Explication |
|---|---|
| ***AllowDBNull*** | Indique si la colonne peut contenir la valeur NULL. |
| ***AutoIncrement*** | Indique si la colonne est auto incrément. |
| ***AutoIncrementSeed*** | Indique la valeur de début pour une colonne auto incrément. |
| ***AutoIncrementStep*** | Indique la pas pour une colonne auto incrément |

| | |
|---|---|
| ***Caption*** | Indique le titre d'entête de la colonne. |
| ***ColumnName*** | Indique le nom de la colonne dans la collection de colonne |
| ***DataType*** | Indique le type de la colonne. |
| ***DefaultValue*** | Indique la valeur par défaut de la colonne. |
| ***Expression*** | Définit l'expression utilisée pour évaluer la valeur de cette colonne (colonne calculée). Expression est une opération arithmétique valide, ou une fonction d'agrégation |
| ***MaxLength*** | Indique le nombre maximum de caractère de la colonne. |
| ***ReadOnly*** | Indique si la colonne est en lecture seulement. |
| ***Table*** | Indique le DataTable auquel le DataColumn appartient. |
| ***Unique*** | Indique si la valeur de cette colonne doit être unique pour l'ensemble des lignes. |

L'objet DataColumn possède plusieurs constructeurs dont les principaux sont

- ***DataColumn()***
- ***DataColumn****(Column_Name, data_Type, expression)*
- ***DataColumn****(Column_Name, data_Type)*

Pour définir le type d'une colonne, on utilise l'instruction ***Type.GetType****(String Type_Name)****( voir III.2)***

## *III Les objets DataTable et DataRow*

### *III.1 Présentation*

L'objet DataTable est un objet analogue à une table dans une base de données, avec des lignes (DataRows) et des colonnes (DataColumns), ainsi que des contraintes.

Un tel objet peut être converti en XML et sérialisé. Les méthodes et propriétés les plus classiques sont :

| **Propriétés ou méthodes** | **Explication** |
|---|---|
| ***CaseSensitive*** | Indique si la comparaison des chaînes dans le DataTable gère la casse. |
| ***ChildRelations*** | Indique la collection de relation fille sur le DataTable |
| ***Columns*** | Représente la collection des colonnes du DataTable. |
| ***Constraints*** | Obtient la collection des contraintes sur le DataTable. |
| ***DataSet*** | Indique le DataTable auquel le DataSet appartient. |
| ***DefaultView*** | Indique la vue par défaut du DataTable. |
| ***PrimaryKey*** | Clé primaire du DataTable. |
| ***Rows*** | Collection de lignes du DataTable. |
| ***TableName*** | Nom de la table dans le DataSet. |
| **AcceptChanges** | Valide toutes les modifications faites sur le DataTable depuis le dernier appel à ***AccepChanges*** |
| ***RejectChanges*** | Annule toutes les modifications faites sur le DataTable depuis le dernier appel à ***AccepChanges***. |
| **Clear** | Supprime toutes les lignes du DataTable. |
| **GetChanges** | ***Surchargé***. Obtient une copie du DataTable contenant toutes le lignes qui ont été modifié depuis le chargement ou l'appel au dernier AcceptChanges. |
| **GetErrors** | Tableau contenant les lignes qui contiennent une erreur. |
| **NewRow** | Crée un DataRow dont le schema est identique à celui du DataTable sans l'ajouter au DataTable. |
| **ReadXml** | ***Surchargé***. Lit les informations d'un fichier Xml et les place dans le DataTable. |
| **Select** | ***Surchargé***. Utilisé pour les filtres. |
| **WriteXml** | ***Surchargé***. Enregistre les informations du DataTable dans un fichier Xml. |

Il importe de signaler que dans une collection ***«col», col[i] représente le ieme élement de la collecton***. De plus, nous avons les méthodes et propriétés ci-dessous :

- ***.Clear() :***Pour supprimer tous les éléments de la collection
- ***.Add(...) :*** Pour ajouter un nouvel élément.
- ***.Count*** qui indique le nombre d'élément dans la collection

Ainsi, si ***table*** est un DataTable,

- ***table.Columns.add(…)*** : permet d'ajouter une nouvelle colonne. Les paramètres sont les mêmes que celles du constructeur de la classe DataColumn
- ***table.Rows[0]*** : indique la i$^{eme}$ ligne du DataTable. C'est une instance de la classe DataRow.
- ***Table.Rows[3].delete()*** permet de supprimer la 3$^{eme}$ ligne colNum.

### *III.2 Manipulation d'un DataTable*

Les différentes lignes d'un **DataTable** sont du type **DataRow**. Un DataRow contient plusieurs versions des données pour un même champ. La version est définie par l'énumération DataRowVersion et peut prendre les valeurs:

- ***Current*** : C'est la valeur auquel on accède par défaut.
- ***Original*** : C'est la valeur originale du champ, quand la table a été créée
- ***Default*** : C'est la valeur par défaut appliquée aux nouvelles lignes.

Si dr est un DataRow, dr[int i], dr[String colName] permet de récupérer ou de modifier la valeur courante de la colonne de rang i, ou de la colonne a pour nom colName. Toute fois, on peut utiliser :

- ***dr[int i, DataRowVersion.Original]*** pour récupérer la valeur original de la colonne i ;
- ***dr[int i, DataRowVersion.Default]*** pour recupérer la valeur par défaut de la colonne i ;

Pour ajouter une ligne à un DataTable, on procède comme suit :

- On appelle dans un premier temps la méthode newRow() du DataTable
- On définit les valeurs du DataRow obtenu
- On ajoute le DataRow modifié à la collection des lignes du DataTable

Un DataRow possède une propriété ***RowState*** permettant d'indiquer l'état de la ligne. Les valeurs possibles de cette propriété sont ***Added***(La ligne a été ajouté et acceptChanges n'a pas encore été appelé), ***Deleted***(la ligne a été supprimé à travers la méthode delete), ***Modified***(La ligne a été modifiée), ***Unchanged***(La ligne n'a subit aucune modification)

Pour parcourir les lignes d'un DataTable, on utilise l'un des codes :

```
foreach (DataRow dr in table.Rows){
        //Traitement du DataRow dr
}
```

```
foreach (int i=0 ;i< table.Rows.count ;i++){
        DataRow dr =table.Rows[i] ;
        //Traitement du DataRow dr
}
```

Le même principe s'applique à la collection des colonnes d'un DataTable.

***Exemple :*** Créer le DataTable de EXPEDITION(**Numero**, principal, frais). Ajoute deux colonnes, l'une qui calcule le montant à payer et le second qui calcule le total des montants. Ajouter deux Expéditions, afficher la liste des expéditions supprimer toutes les celles dont les frais sont nuls, enregistrer les informations dans le fichier Xml.

```
//Création des colonnes
        DataColumn colNum = new DataColumn();
        colNum.ColumnName = "Numero";                          colNum.Caption = "N°";
        colNum.DataType = Type.GetType("System.Double");       colNum.AutoIncrement = true;
        DataColumn colPrincipal= new DataColumn("principal", Type.GetType("System.Double"));
        DataColumn colFrais= new DataColumn("frais", Type.GetType("System.Double"));
        DataColumn colMt= new DataColumn("montant", Type.GetType("System.Double"), "principal+frais");

//Création du DataTable
        DataTable tbl_Expedition=new DataTable() ;          tbl_Expedition .TableName= "EXPEDITION" ;
```

```
//Ajout des colonnes au DataTable
        tbl_Expedition.Columns.add(colNum) ;                    tbl_Expedition.Columns.add(colPrincipal) ;
        tbl_Expedition.Columns.add(colfrais) ;                  tbl_Expedition.Columns.add(colMt) ;
        tbl_Expedition.Columns.add("total", Type.GetType("System.Double"), "sum(montant)");

 //Définition de la clé primaire
         tbl_Expedition.PrimaryKey = new DataColumn[] { colNum };

//Ajout des informations
         DataRow dr = tbl_Expedition.NewRow();
          dr["principal"] = 20000;                          dr["frais"] = 700;        tbl_Expedition.Rows.Add(dr);
         dr = tbl_Expedition.NewRow();
          dr["principal"] = 40000;                dr["frais"] = 2000;               tbl_Expedition.Rows.Add(dr);

//Affichage des informations
        foreach (DataRow dr1 in tbl_Expedition.Rows){
                for (int i = 0; i < tbl_Expedition.Columns.Count; i++)      Console.Write(dr1[i]);
                 Console.Write("\n");
        }

//suppression des informations
        foreach (DataRow dr1 in tbl_Expedition.Select("frais=0"]))        dr1.delete() ;

//Enregistrement dans le fichier Xml
        tbl_Expedition.WriteXml(("c:/info.xml")
```

## IV Les objets DataSet, DataRelation et Constraint

### IV.1 L'objet DataSet

Un DataSet est ensemble de DataTable muni d'un ensemble de relation et de contraintes. Les tables d'un DataSet peuvent être indéxés à partir de leurs index (de base 0) ou leurs noms. Les informations d'un DataSet sont gérées à travers un fichier Xml.

Les méthodes et propriétés de cet objet sont :

- ***.Tables :*** Pour obtenir la collection des tables du DataSet
- ***.Tables.add( Nom) :*** Pour ajouter une table
- ***.writeXml(…) :*** Pour écrire dans un fichier Xml
- ***.readXml(…) :*** Pour lire d'un fichier Xml
- ***.Relation :*** Pour obtenir la collection de relation.

### IV.2 Les contraintes

Il y a deux sortes de contraintes dans ADO.NET: ForeignKeyConstraint et UniqueConstraint. Une contrainte d'unicité est crée sur une table, alors qu'une contrainte de clé étrangère est créée entre plusieurs tables.

#### IV.2.1 L'objet UniqueConstraint

Elle permet de créer une contrainte d'unicité sur une table. Cette contrainte peut être créée de manière automatique en définisant à ***true***, la propriété ***Unique*** d'un DataColumn, ou en définissant la clé primaire de la relation. Elle est très utile dans le cadre des clés candidates.

Il existe plusieurs constructeurs pour cette classe, donc :

- ***UniqueConstraint(DataColumn) :*** Crée une contrainte d'unicité sur la colonne indiquée
- ***UniqueConstraint(DataColumn, Boolean) :*** Crée un contrainte d'unicité sur la colonne. Le booléen indique si la colonne est la clé primaire
- ***UniqueConstraint(String, DataColumn, Boolean) :*** Crée un contrainte d'unicité sur la colonne. Le booléen indique si la colonne est la clé primaire, et la chaine est le nom de la contrainte.

Les contraintes ci-dessus peuvent s'appliquer à un tableau de colonne.

Une fois crée, la contrainte doit être ajoutée à la collection des colonnes du DataTable.

### IV.2.2 L'objet ForeignKeyConstraint

***ForeignKeyConstraint*** applique des règles sur la manière dont les mises à jour et les suppressions dans les tables connexes sont propagées. Cet objet possède plusieurs constructeurs permettant de définir une contrainte de clé étrangère. Les principales sont :

Il faut par la suite ajouter cette contrainte au DataSet.

- ***ForeignKeyConstraint (DataColumn colonneParent, DataColumn colonneFille) :*** Créée un contrôle de clé étrangère entre les deux colonnes.
- ***ForeignKeyConstraint (String nom, DataColumn colonneParent, DataColumn colonneFille)***

Les contraintes ci-dessus peuvent s'appliquer à un tableau de colonne.

Une fois crée, la contrainte doit être ajoutée à la collection des contraintes du DataTable qui contient la colonne fille.

Un objet ForeignKeyConstraint possède en plus les propriétés ***DeleteRule*** et ***UpdateRule*** qu'il faut utiliser en cas de suppression et de mise à jour. Les valeurs de ces propriétés sont : ***Rule.Cascade, Rule.None, ….***

La contrainte ne sera prise en compte que lorsque la propriété ***EnforceConstraints*** du DataSet est positionnée à true.

### IV.2.3 L'objet DataRelation

Cet objet permet de décrire les relations existantes dans un DataSet, pour vous déplacer dans les tables et pour retourner les lignes enfants ou parentes d'une table associée. Le fait d'ajouter un DataRelation à un objet DataSet ajoute, par défaut, un objet UniqueConstraint à la table parente et un objet ForeignKeyConstraint à la table enfant. Toutefois, la création d'un DataRelation n'implique pas automatiquement celle d'un ForeignKeyConstraint, c'est en ce sens qu'un DataRelation est différente d'un ForeignKeyConstraint.

La classe DataRelation possède les constructeurs :

- ***DataRelation(String nom, DataColumn colonneParent, DataColumn colonneFille):*** Crée la Relation et ajoute une contrainte de clé étrangère
- ***DataRelation(String nom, DataColumn colonneParent, DataColumn colonneFille, boolean ):*** Crée la Relation et ajoute une contrainte de clé étrangère si le booléen est fixée à true.

Les contraintes ci-dessus peuvent s'appliquer à un tableau de colonne.

Une fois crée, la contrainte doit être ajoutée à la collection des Relations du DataSet.

La classe DataRow possède les méthodes :

- ***GetChildRows(String nomRelation):*** Qui renvoie les lignes filles liées à cette colonne à travers la relation.
- ***GetParentRows(String nomRelation):*** Qui renvoie les lignes parentes liées à cette colonne à travers la relation.

```
//Création des DataTables
        DataTable Etudiant=new DataTable(" Etudiant")
        DataTable Inscription=new DataTable("Inscription") ;
//Création du DataSet
        DataSet ds=new DataSet() ;      ds.Tables.add(Etudiant) ;ds.Tables.add(Inscription)

//Définition des colonnes
        etudiant.Columns.add("mle", Type.GetType("System.String"));
        etudiant.Columns.add("nom", Type.GetType("System.String"));
        etudiant.Columns.add("sexe", Type.GetType("System.String"));

        Inscription.Columns.add("Numero", Type.GetType("System.Double"));
        Inscription.Columns.add("mle", Type.GetType("System.String"));
        Inscription.Columns.add("annee", Type.GetType("System.String"));
        Inscription.Columns.add("sexe", Type.GetType("System.String"));

//Clé primaire de la table Inscription
                tbl_Inscription.PrimaryKey = new DataColumn[] { Inscription.Columns["mle"]};
```

```
//Définition d'une contrainte d'unicité : on ne peut prendre deux inscription au cours de la même année.
        Columns Col=new Columns[]{Inscription.Columns["mle"], Inscription.Columns["annee"]} ;
        UniqueConstraint  c1= new UniqueConstraint("UK_INSC", col)              Etudiant.Constraints.Add(c1) ;
//Création d'une relation entre etudiant et Inscription sur le mle
        Column colPere= Etudiant.Columns["mle"] ;       Column colMere= Inscription.Columns["mle"]
        DataRelation  dr= new DataRelation("FK_Etudiant_Inscription_mle", colPere,colFille)
        ds.Relations.add(dr) ;
        dr.ChildKeyConstraint.UpdateRule=Rule.Cascade ;
//Affichage des informations
        foreach (DataRow dr1 in etudiant.Rows){
                foreach (DataRow dr2 in dr1.GetChildRows("FK_Etudiant_Inscription_mle"){
                        //Traiter dr2
                }
        }
```

## V L'objet DataAdapter

*Lorsqu'on travaille avec une BD, on ne construit pas manuellement nos DataTables et DataSet. En fait ces objets, sont une copie des tables de la BD.*

*Cet objet permet de faire le lien entre les objets du mode connecté et ceux du mode déconnecté. Il est utilisé pour charger un DataSet ou DataTable à partir d'une ou plusieurs tables de la BD, de mettre à jour les* modifications du DataTable ou DataSet au sein de la source de données.

Les propriétés et méthodes de cet objet sont:

| ***Propriétés ou méthodes*** | **Explication** |
|---|---|
| ***DeleteCommand*** | Commande utilisée pour supprimer les données qui ont été supprimées |
| ***InsertCommand*** | Commande utilisée pour insérer les données ajoutées |
| ***SelectCommand*** | Commande utilisée pour recupérer les informations de la BD |
| ***UpdateCommand*** | Commande utilisée pour mettre à jour les données modifiées |
| **MissingSchemaAction** | Indique l'action a effectuer si les informations sur le schema sont absente. |
| **TableMappings** | Indique comment se fera le mappage entre les différents éléments de la commande et le DataSet. |
| ***AcceptChangesDuringUpdate*** | Indique si l'appel de la méthode fill provoque celle de AcceptChages du DataSet ou DataTable |
| ***Fill*** | Ajoute ou modifie les données de sorte à être en concordance avec celui de la source de données |
| ***FillSchema*** | Met à jour le schéma des tables |
| ***Update*** | Met à jour la source de données sur la base des modifications  dans la copie locale. |

### V.1 Remplissage d'un DataTable à partir d'une requête

Le remplissage d'un DataSet ou DataTable se fait via un DataAdapter et sa méthode ***fill.*** Le code est de la forme :

```
XYAdapter  adp =new XYAdapter("Requete", "chaine_connexion") ;
DataTable dt=new DataTable() ;
Adp.fill(dt);
```

Le principe est le suivant :

- La méthode fill ouvre la connexion définit au niveau du DataAdapter
- L'objet DataAdapter effectue la requête et initialise le DataTable avec un schema correspondant à celui de la requête.
- Les lignes de la requête sont insérrés dans le DataTable.

***Remarque :***

La méthode **Fill** du **DataAdapter** remplit un objet DataSet uniquement avec les colonnes et les lignes de la table d'une source de données ; bien que les contraintes soient généralement définies par la source de données, la méthode **Fill** n'ajoute pas ces informations de schéma au **DataSet** par défaut.

Pour remplir un **DataSet** avec les informations de contrainte de clé primaire existantes provenant d'une source de données, vous pouvez :

- appeler la méthode **FillSchema** du **DataAdapter**
- affecter **AddWithKey** à la propriété **MissingSchemaAction** du **DataAdapter** avant d'appeler **Fill**.

Cela garantira que les contraintes de clé primaire dans le **DataSet** reflètent celles de la source de données. Les informations de contrainte de clé étrangère ne sont pas incluses et doivent être explicitement créées.

## VI Remplir une table à partir d'une requête et mettre à jour les modifications dans la source de données.

### *VI.1 Mettre à jour les modifications dans la source de données*

Il est à noter que les modifications effectuées dans un DataTable sont différentes e celles effectuées dans la source de données. Ils le sont uniquement en mémoire il faut les répercuter dans la source de données.

Pour insérer, modifier, supprimer les lignes du DataTable au sein de la source de données il faut faire appel à la méthode ***update*** du DataAdapter. Cette méthode demande à ce que 04 propriétés soient définies au préalable:

- ***SelectCommand:*** contient la commande ayant permis de construire le DataTable
- ***InsertCommand***: définit la commande permettant d'ajouter une nouvelle ligne.
- ***UpdateCommand:*** définit la commande permettant de modifier une nouvelle ligne.
- ***DeleteCommand:*** définit la commande permettant de supprimer une nouvelle ligne.

Les 3 dernières propriétés peuvent être définies de manière automatique en utilisant un objet CommandBuilder. On obtient alors le schema :

```
XYDataAdapter adp =new XYDataAdapter("Requete", "chaine_connexion") ;
DataTable dt=new DataTable() ;
Adp.fill(dt);
//Ajout et suppression dans le DataSet et DataTable
…

//'Enregistrement des données
XYCommandBuilder bld  =new XYCommandBuilder(adp) ;
Adp.update(dt);
```

*Exemple:*

Considérons une BD ACCESS( scolaire.mdb) possédant la table suivante:

Filiere(***codeFil***, libelleFil)

Specialite(***codeS***,libelles, codeFil, inscription, scolarite),

Écrire du code pour ajouter la spécialité IG, supprimer les spécialités dont l'inscription est nulle et ajouter de 10% le montant de l'inscription.

On obtient le bout de code :

☞

```
OleDbConnection con= new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=scolaire.mdb;
");
OledbDataAdapter adp= new OledbDataAdapter("Select * from filiere;Select * from Specialite", con);
DataSet ds=new DataSet();
//Remplissage du DataSet
```

```
        adapter.MissingSchemaAction= MissingSchemaAction.AddWithKey;
        adapter.TableMappings.Add("TblFiliere", "Filiere");
       adapter.TableMappings.Add("TblSpecialite", "Specialite");
       adapter.Fill(ds);

  //Ajout d'une colonne calculé
              ds.Tables["TblSpecialite"].Columns.add("pension",Type.GetType("System.Double"),"inscription +
scolarite")
  //Ajout d'une ligne
     DataRow dr= ds.Tables["TblSpecialite"].newRow();
     dr["code"]="IG" ;                    dr["Libelle"]="Informatique de Gestion"
     dr["inscription"]=20000 ;            dr["scolarite"]=300000 ;              dt.rows.add(dr)
  //suppression des lignes
     DataRow [] rowsToDelete = ds.Tables["TblSpecialite"].select("inscription=0")
     foreach(DataRow  x in rowsToDelete)      x.delete()
  //modification des lignes
     foreach(DataRow  x in ds.Tables["TblSpecialite"].Rows)     x["inscription"]=x["inscription"]*1.1
  //Mise à jour des données
     OledbCommandBuilder bld=new OledbCommandBuilder(adp) ;
     Adp.update(ds);
```

On aurait pu écrire le même code en utilisant deux objets DataAdapter.

## *VII L'objet DataView*

Un DataView permet d'avoir une « vue » d'une DataTable. Cela permet aussi de pouvoir la trier, la modifier. La fonction principale de cet objet est de relier une base de données aux applications WinForm et WebForm.

Cet objet dispose des proprieties et methods ci-dessous:

| ***Propriétés ou méthodes*** | **Explication** |
|---|---|
| **RowFilter** | Définit la condition utilisée pour filtrer les lignes. |
| **Sort** | Définit la condition de tri |
| **Count** | Indique le nombre de lignes dans le DataView |
| ***addNew()*** | Ajoute une nouvelle ligne dans le DataView |
| ***Delete (int i)*** | Supprime la ligne se trouvant à l'indice i |

# CHAPITRE IV LIAISON DES CONTROLES GRAPHIQUES A UNE SOURCE DE DONNEES et ASP.NET

Pour interagir efficacement avec un utilisateur, les données contenues dans la BD doivent être mis à la disposition de l'utilisateur pour consultation, mise à jour et suppression. Sous ADO.NET, plusieurs classes permettent de contrôler cette interaction. Dans cette liste, on peut lister les classes :

- ***BindingSource***
- ***DataBinding***

## I La classe BindingSource

Le contrôle ***BindingSource*** permet de simplifier la liaison des contrôles à une source de données. Les propriétés et méthodes liées à cet objet sont:

| *Propriétés ou méthodes* | **Explication** |
|---|---|
| **RowFilter** | Définit la condition utilisée pour filtrer les lignes. |
| **Sort** | Définit la colonne de tri |
| **Filter** | Définit la condition utilisée pour filtrer les lignes. |
| **DataSource** | Indique la source de données auquel l'objet puise ses données. Il peut s'agir d'un DataSet, un DataTable, … |
| **Position** | Indique la position de l'élément courant dans la liste |
| **DataMember** | Indique la liste spécifique dans laquelle l'objet puise ses données. Il peut s'agir d'une relation, d'une colonne. |
| **Current** | Indique l'élément courant dans le BindingSource |
| ***addNew()*** | Ajoute une nouvelle ligne dans le DataView |
| ***EndEdit()*** | Applique les modifications du DataBinding dans la source de données. |
| ***Delete (int i)*** | Supprime la ligne se trouvant à l'indice i |
| ***RemoveCurrent()*** | Supprime l'élément courant de la liste |
| ***RemoveAt(int i)*** | Retire l'élément qui se trouve à l'indice i |

## II La classe Binding

Le ***DataBinding*** consiste à prendre les données d'une source que l'on appelle «provider»(DataTable, DataView, BindingSource, …) et de les placer par simple appel de méthode dans un contrôle graphique appelé «Consumer »(TextBox, DataGrid, DropDownList, …).

La relation entre le provider et le consumer est appelée «Binding».

Lorsqu'un contrôle est capable d'afficher plusieurs valeurs, la liaison entre le consumer et le provider se fait à travers la propriété DataSource du consumer. C'est le cas du contrôle DataGridView.

Ainsi pour lier un DataGridView «dg» à une source de données «dataSource», on utilise le boût de code :

***dg.DataSource=dg ;***

En revanche pour les contrôles ne pouvant afficher plusieurs valeurs, cette opération est complexe et nécessite un objet Binding.

Dans le cas d'une liaison à un contrôle graphique, le constructeur les plus classiques est :

***Binding(String** propertyName**, Object dataSource, String** column**)***

Où :

- ***Property*** : spécifié la propriété du contrôle qui sera utilisé pour la liaison. C'est généralement la propriété ***Text*** qui est utilisé dans le cadre d'une liaison.
- ***dataSource*** : Représente la source de données

- ***column***: Indique la colonne du DataSource qui sera utilisé pour alimenter la propriété du contrôle.

Pour ajouter une liaison à un contrôle, on utilise la propriété **DataBindings** du contrôle. Cette propriété représente la collection des liaisons associées au contrôle.

Exemple :

```
Binding bmle = new Binding("Text",dataEmploye, "mle");
Binding blnsc=new Binding("Text", dataEmploye, "scolarite", true, DataSourceUpdateMode.OnPropertyChanged, null, "#,##0";);
txtMatricule.DataBindings.Add(bmle);
txtInscription.DataBindings.Add(blnsc);
```

Dans le cas des contrôles ComboBox, il faut indiquer en plus la source dans laquelle elle puisse les données qui seront affichée dans la liste déroulante du ComboBox.

```
ComboBox cb;
cb.DataSource=Source_donnee_affiché;
cb.DisplayMember=champ_affiche ;
cb.ValueMember=champ_contenant_valeur_associe ;
```

La liason peut alors se faire avec la propriété selectedValue ou selectedText, ou Text

On peut faire usage de la propriété DataMember pour indiquer un sous élément de la source de données. C'est le cas lorsqu'on travaille avec les relations.

**Travail Pratique :**

- DataGridViewComboBoxColumn et ses dérivées
- Filtrage des informations
- Utilisation de ASP.NET

## *III Résumé*

Cet article explique étape par étape comment récupérer par programmation des données d'une base de données Microsoft Access puis les présenter à l'utilisateur. L'objet de cet article n'est pas de fournir toutes les solutions possibles à ce problème. Il propose une solution simplifiée qui utilise ASP.NET, ADO.NET et Visual C# .NET comme introduction à des technologies connexes. Microsoft Windows 2000 Professionnel, Windows 2000 Server, Windows 2000 Advanced Server ou Windows Server 2003

- Microsoft Internet Information Services (IIS)
- Microsoft .NET Framework 1.0 ou Microsoft .NET Framework 1.1
- Base de données Les Comptoirs Microsoft Access

### *III.1 Créer une application Web ASP.NET à l'aide de Visual C# .NET*

# TP 1 ADO.NET / Connexion à la BD et exécution des requêtes

Il est question dans ce TP d'écrire une application console permettant de se connecter à une BD et d'exécuter requêtes. Les étapes ci-dessous vous permettront d'écrire ledit programme :

Partie **1 : Création de la BD sous MYSQL**

1) Créer une nouvelle BD sous MYSQL
2) Ajouter à cette BD la table ci-dessous :
   Specialite(**codeSpec**, libelleSpec, inscription, scolarite))

3) Insérez dans chacune de vos tables au moins 02 enregistrements.

**Partie 2 : Accès à la BD**

1) Créer un nouveau programme Console c#
2) Quels sont les fournisseurs d'accès pour une BD MYSQL ?

Dans la suite du cours on utilisera le fournisseurs Mysql. Pour utiliser ce fournisseur, il faut l'installer au préalable sur votre machine.

3) Installer ( si ce n'est pas encore fait) sur votre machine le fournisseur d'accès de mysql qui se trouve dans le répertoire de ressource(fichier : mysql-connector-net-6.6.5). Prenez le soin de faire une installation complète. Pour l'intégration de MYSQL à VS2012, Installer par la même occasion ***mysql-installer-community-5.6.10.0*** en sélectionnant les outils ***Mysql Connector***

Pour utiliser les classes de MYSQL, il faut ajouter la référence à MYsql.data à votre projet.

4) En utilisant le menu ***projet→ajouter une reference***, ajouter une référence à Mysql.Data à votre projet
5) Dans la classe Program, ajouter une méthode seConnecter qui permet de se connecter à la BD (n'oublier de faire référence au namespace Mysql.data.mysqlClient)

```
String chaine_connexion=".....";
XYConnection con =new XYConnection(chaine_connexion)
con.open( )
```

6) Ecrire une méthode qui affiche la liste des spécialités(DataReader)
7) Ecrire une méthode qui prend en entrée le code, le libelle, l'inscription et la scolarité d'une spécialité et l'enregistre dans la BD (requêtes paramétrées)
8) Ecrire une méthode qui renvoie le nombre de Specialite dont l'inscription est supérieure à 100000
9) Faites appel à ces méthodes dans la méthode main.
10) Transformer votre projet en projet WindowsForms
11) Créer un formulaire qui permet de créer une spécialité
12) Créer un formulaire qui demande à un utilisateur qui demande à un utilisateur le code d'une spécialité et affiche le libelle et l'inscription de cette spécialité.

**Partie 3 : Généralisation**

1) Modifier le programme précédent en considérant que la BD est une BD ACCESS, Vous pouvez utiliser le menu ***outils→Connexion à la base de données*** pour obtenir la chaine de connexion.
2) La résolution de la question ci-dessus implique de nombreuses modifications dans le code. En utilisant lors des déclarations les objets de la classe System.Data***(System.Data.IDbConnection, System.Data.IDbCommand)*** montrer qu'on peut changer de SGBD sans avoir à trop modifier le code.

Partie 4 : ASP.NET

Utiliser le texte ci dessous pour réaliser une application ASP.NET

1. Démarrez Microsoft Visual Studio .NET.
2. Dans le menu **Fichier**, pointez sur **Nouveau**, puis cliquez sur **Projet**.
3. Dans la boîte de dialogue **Nouveau projet**, cliquez sur **Projets Visual C#** sous **Types de projet**, puis cliquez sur **Application Web ASP.NET** sous **Modèles**.
4. Dans la zone de texte **Emplacement**, remplacez le nom par défaut **WebApplication #** le votre

### *Création de l'exemple de formulaire Web*

L'exemple de code dans cette section utilise un contrôle de serveur ASP.NET de table pour générer de façon dynamique une présentation simplifiée des données extraites. ASP.NET propose différents contrôles flexibles que vous pouvez utiliser pour fournir d'autres méthodes de restitution des données. Pour plus d'informations sur les contrôles pris en charge par ASP.NET, consultez la section RÉFÉRENCES à la fin de cet article.

Ajoutez un nouveau formulaire Web nommé DataSample.aspx à votre application Web ASP.NET dans Visual Studio .NET. Pour cela, procédez comme suit :

a. Dans l'Explorateur de solutions, cliquez avec le bouton droit sur le nœud du projet, cliquez sur **Ajouter**, puis sur **Ajouter un formulaire Web**.
b. Dans la zone de texte **Nom**, tapez DataSample.aspx, puis cliquez sur **Ouvrir**.

À partir de la boîte à outils **Formulaires Web**, faites glisser un contrôle **Table** vers la page .aspx en mode Création.

Dans **Propriétés**, remplacez **ID** par **DisplayTable**.

Dans l'Explorateur de solutions, cliquez avec le bouton droit sur la page .aspx, puis cliquez sur **Afficher le code**.

Ajoutez la référence d'espace de noms suivante en début du fichier de classe code-behind :

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

Remplacez le gestionnaire d'événement **Page_Load** par le code suivant :

```
protected void Page_Load(object sender, EventArgs e)
{
    String connectString = "Database=prj_ges_scolaire;Data Source=localhost;User Id=root;Password=";
    //Create de la connection
    MySqlConnection cn = new MySqlConnection(connectString);

    //Open the connection.
    cn.Open();
    //Use a variable to hold the SQL statement.
    string selectString = "SELECT codeSpec, libspec, codefil FROM specialite";

    MySqlCommand cmd = new MySqlCommand(selectString, cn);

    MySqlDataReader reader = cmd.ExecuteReader();

    //Set a table width.
    DisplayTable.Width = Unit.Percentage(90.00);
    //Create a new row for adding a table heading.
    TableRow tableHeading = new TableRow();

    TableHeaderCell codeSpecHeading = new TableHeaderCell();
    codeSpecHeading.Text = "Code specialité";
```

```
codeSpecHeading.HorizontalAlign = HorizontalAlign.Left;
tableHeading.Cells.Add(codeSpecHeading);

TableHeaderCell SpecLibHeading = new TableHeaderCell();
SpecLibHeading.Text = "Libellé";
SpecLibHeading.HorizontalAlign = HorizontalAlign.Left;
tableHeading.Cells.Add(SpecLibHeading);

TableHeaderCell filHeading = new TableHeaderCell();
filHeading.Text = "Filière";
filHeading.HorizontalAlign = HorizontalAlign.Left;
tableHeading.Cells.Add(filHeading);

DisplayTable.Rows.Add(tableHeading);

//Loop through the resultant data selection and add the data value. for each respective column in the table.
while (reader.Read())
{
    TableRow detailsRow = new TableRow();
    TableCell customerIDCell = new TableCell();
    customerIDCell.Text = reader["codespec"].ToString();
    detailsRow.Cells.Add(customerIDCell);

    TableCell contactNameCell = new TableCell();
    contactNameCell.Text = reader["libspec"].ToString();
    detailsRow.Cells.Add(contactNameCell);

    TableCell filCell = new TableCell();
    filCell.Text = reader["codefil"].ToString();
    detailsRow.Cells.Add(filCell);

    DisplayTable.Rows.Add(detailsRow);


}
```

Modifiez la variable **connectString** située en début du code afin qu'elle pointe sur l'emplacement de votre base de données Les Comptoirs.

Dans le menu **Fichier,** cliquez sur **Enregistrer tout** afin d'enregistrer le Formulaire Web et les fichiers projet associés.

Dans le menu **Générer,** cliquez sur **Générer la solution** pour générer le projet.