

# 420-KBA-LG, programmation de bases de données

Saliha Yacoub

- › Rappels:
  - L'objet SqlConnection
    - › Propriétés
    - › méthodes
  - L'objet SqlCommand
    - › Propriétés
    - › Méthodes
  - L'objet SqlDataReader
    - › Propriétés
    - › Méthodes.
- › L'objet SqlParameter
  - Retour sur SqlCommand
  - Cas de paramètres en IN
  - Cas d'une fonction stockée.

# ADO.NET

## Introduction:

- › Il est rare que l'exploitation des bases de données se fasse directement par l'interface client des SGBD (Sqldeveloper, SSMS, MySQL Workbench).
- › Les développeurs conçoivent des applications afin de rendre l'accès au données facile et convivial.
- › Il existe au moins deux solutions pour d'accéder aux données d'une bases de données :

1 - Une solution offerte par les SGBDs: les objets permettant l'accès aux données est du côté du SGBD (Exemple: Mysqli, Oracle Call Interface). Dans ce cas les programme C# qui va accéder à un BD SQL Server et à une BD Oracle sont très différents. Les solutions ne sont pas portables d'un SGBD à l'autre. Par contre les accès aux données sont très rapides.

# ADO.NET

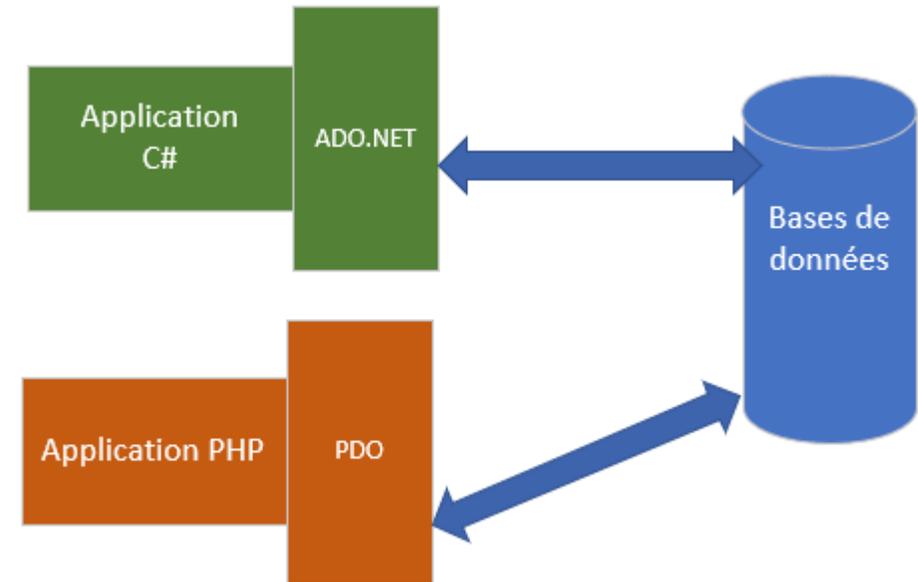
2- Une solution offerte par les langages de programmation. (ADO.NET,PDO,JDBC).

Dans ce cas , la solution est portable d'un SGBD à un autre. L'accès par C# à une BD SQL Server ou une BD MySQL est presque le même.

La figure ci-après montre les accès côté application à une BD.

Une application C#, par ADO.NET

Une application PHP par PDO.



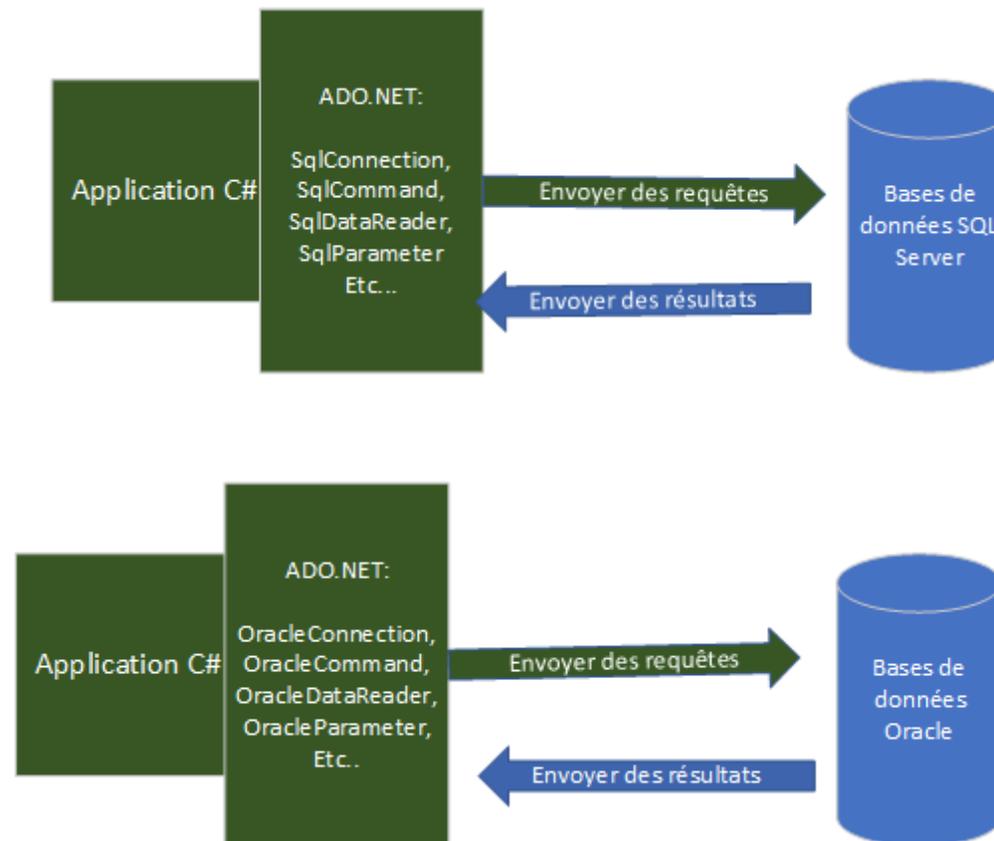
# ADO.NET

## Définition

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.

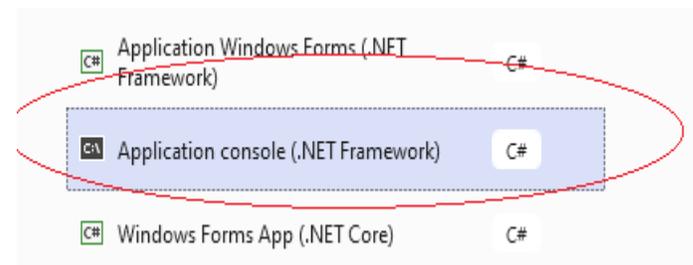
Dans une application ADO.NET, pour accéder à SQL server ou à Oracle, seul le nom des classes change. Les propriétés et les méthodes restent les même



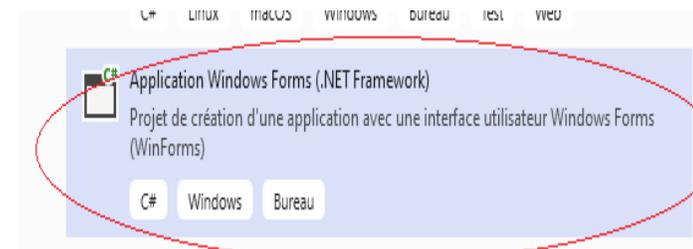
# ADO.NET

Type d'application C#:

En mode consoles



En mode graphique



En tout temps, utiliser le  
using System.Data.SqlClient

```
4 using System.Text;  
5 using System.Threading.Tasks;  
6 using System.Data.SqlClient;  
7
```

# ADO.NET

## La classe SqlConnection: Définition

- › 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()**

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

Pour se connecter à la base de données, nous avons besoin:

- De fournir les informations de connexion (nom du serveur, nom de la bd, nom du user, le mot de passe). Ces informations de connexion formeront une chaîne de connexion.
- Ouvrir une connexion.

# ADO.NET

## La classe SqlConnection: Propriétés importantes:

### 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 : (séparés par un point virgule)

**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

```
string chaineConnexion = "Data source =M-INFO-SY\\SQLEXPRESS; Initial Catalog =EmpClgDB; User Id= Patoche;password =123456";
```

# ADO.NET

## **La classe SqlConnection: Propriétés importantes:**

À ce stade nous avons un objet SqlConnection: connSql

Nous avons une chaîne de connexion: chaineConnexion.

La propriété ConnectionString va permettre de passer la chaîne de connexion.

```
connSql.ConnectionString = ChaineConnexion;
```

# ADO.NET

## La classe SqlConnection: Propriétés importantes:

### 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

```
connSql.State;
```

## La classe SqlConnection: méthodes importantes:

Il existe deux méthodes importantes pour cette classe:

Open() → permet d'ouvrir une connexion

```
connSql.Open();
```

Close() → permet de fermer une connexion ouverte.

```
connSql.Close();
```

# ADO.NET

```
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= user1;password =123";
                connSql.ConnectionString = chaineConnexion;
                connSql.Open();
                Console.WriteLine("statut de la connexion" + " " + connSql.State);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message) }
            connSql.Close();
            Console.WriteLine("statut de la connexion" + " " + connSql.State);
        }
    }
}
```

## La classe SqlCommand: Propriétés importantes:

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**).

SqlCommand possède plusieurs constructeurs, pour commencer, celui que nous allons utiliser est le constructeur suivant :

Un objet SqlCommand se crée avec la méthode new();

```
SqlCommand(string SQL , SqlConnection connSql)
```

```
SqlCommand sqlComd = new SqlCommand(sql, connSql);
```

sql est une requête SQL

connSql est un objet SqlConnection

# ADO.NET

## La classe SqlCommand: Propriétés importantes:

Propriétés	Signification
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 stockée).
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

Par défaut la propriété CommandType n'est pas une procédure

```
sqlCmd.CommandType = CommandType.StoredProcedure
```

SQL qui

## La classe SqlCommand: méthodes importantes

Méthodes	Significations
<b>ExecuteNonQuery()</b>	Exécute une instruction SQL sur Connection et retourne le nombre de lignes affectées. Pour toutes les requêtes DML
<b>ExecuteReader()</b>	Surchargé. Envoie CommandText à Connection et génère SqlDataReader. Pour les requêtes SELECT
<b>ExecuteScalar()</b>	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...

# ADO.NET

nous sommes déjà connectés: Le nom de l'objet SqlConnection est: connSql

## Exemple1: Cas d'une instruction DML (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();
```

## Exemple 2: Cas d'un SELECT qui retourne UNE SEULE valeur (ici un INT)

```
string sql3 = "select count(*) from employesClg";  
  
SqlCommand sqlCommdCount = new SqlCommand(sql3, connSql);  
  
int totalEmp = (int)sqlCommdCount.ExecuteScalar();  
  
Console.WriteLine(totalEmp);  
  
Console.Read();
```

# ADO.NET

## La classe SqlDataReader: définition :

un objet **DataReader** sert 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). Le contenu d'un SqlDataReader correspond à une requête SELECT (plusieurs lignes, plusieurs colonnes. En ce sens c'est comme un curseur).

L'objet SqlDataReader NE se crée PAS avec la Méthode new()

L'objet SqlDataReader se crée avec la méthode ExecuteReader() de l'objet SqlCommand

```
SqlCommand sqlComd = new SqlCommand(sql, connSql);  
SqlDataReader sqlRead = sqlComd.ExecuteReader();
```

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 indiquée par 0 est de type **int**, alors il est possible de la récupérer à l'aide de la méthode **GetInt64(0)** de l'objet **DataReader**.

## La classe SqlDataReader: méthodes importantes

Méthodes	Signification
Close()	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.

## La classe SqlDataReader: Exemple

```
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();
```

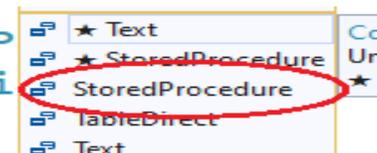
# ADO.NET, SqlParameter

## La classe SqlCommand: Propriétés importantes:

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

Par défaut la propriété CommandType a la valeur Text (pour une requête SQL qui n'est pas une procédure

```
SqlParameter paramCategorie = new SqlParameter("Categorie", SqlDbType.Text);  
paramCategorie.Direction = ParameterDirection.Input;
```



## La classe SqlParameter: Constructeurs:

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

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

# ADO.NET

Direction du paramètre:

Paramètre de la procédure	Direction du SqlParameter
IN(par défaut)	Input (par défaut)
OUT	Output
Une fonction	ReturnValue

```
SqlParameter categorie = new SqlParameter("@categorie", Sq  
categorie.Direction = ParameterDirection.;
```

- ★ Input
- ★ ReturnValue
- ★ Output
- ★ InputOutput
- Input
- InputOutput
- Output

## › La classe SqlParameter: Propriétés importantes

<b>Direction</b>	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.
<b>ParameterName</b>	Obtient ou définit le nom de SqlParameter
<b>SqlDbType</b>	Spécifie le type de données Sql
<b>Size</b>	Obtient ou définit la taille maximale, en octets, des données figurant dans la colonne.
<b>Value</b>	Obtient ou définit la valeur du paramètre

# ADO.NET

- › Comment passer une procédure et ses paramètres ?
  - Étape 0: Le SqlCommand va permettre d'indiquer clairement qu'il s'agit d'une procédure par ses propriétés CommandText et CommandType:

```
SqlCommand cmdInsert = new SqlCommand("ajouterJoueur", sqlconn);  
cmdInsert.CommandText = "ajouterJoueur";  
cmdInsert.CommandType = CommandType.StoredProcedure;
```

**ajouterJoueur** est le nom de la procédure stockée

- › Comment passer une procédure et ses paramètres ?
  - Étape 1: définir un SqlParameter pour chaque paramètre de la procédure

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

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

```
SqlParameter parmSalaire = new SqlParameter("@salaire", SqlDbType.Decimal);  
parmSalaire.Direction = ParameterDirection.Input;
```

```
SqlParameter parmEquipe = new SqlParameter("@codeEquipe", SqlDbType.Char, 3);  
parmEquipe.Direction = ParameterDirection.Input;
```

La procédure `ajouterJoueur` a 4 paramètres. Ils sont tous en IN

- › Comment passer une procédure et ses paramètres ?
  - Étape 2: donner des valeurs aux paramètres en utilisant le propriété Value de SqlParameter

```
parmNom.Value = 'Patoche';  
parmPrenom.Value = 'Alain';  
parmSalaire.Value = 100000;  
parmEquipe.Value = 'MTL';
```

Les valeurs des paramètres peuvent être fournis par des zones de texte.

# ADO.NET

- › Comment passer une procédure et ses paramètres ?
  - Étape 3: Ajouter les paramètres avec leurs valeurs à SqlCommand avec la propriété **Parameters** de SqlCommand.

```
cmdInsert.Parameters.Add(parmNom);  
cmdInsert.Parameters.Add(parmPrenom);  
cmdInsert.Parameters.Add(parmSalaire);  
cmdInsert.Parameters.Add(parmEquipe);
```

Les valeurs des paramètres peuvent être fournis par des zones de texte.

# ADO.NET

- › Comment passer une procédure et ses paramètres ?
  - Étape 4: Exécuter la procédure Stockée. Il faudra utiliser la méthode appropriée de SqlCommand.

```
cmdInsert.ExecuteNonQuery();
```

**ajouterJoueur** est le nom de la procédure stockée qui fait une insertion . Donc la méthode appropriée de sqlCommand est `ExecuteNonQuery()`;

# ADO.NET

Exemple de fonction sans paramètre qui retourne une valeur:

```
private void compter()
{
    SqlCommand sqlcmdTotal = new SqlCommand("totalJoueur", sqlconn);
    sqlcmdTotal.CommandText = "totalJoueur";
    sqlcmdTotal.CommandType = CommandType.StoredProcedure;

    //On déclare le paramètre de retour
    SqlParameter resultat = new SqlParameter("@totaljoueur", SqlDbType.Int);
    resultat.Direction = ParameterDirection.ReturnValue;

    //ajouter les paramètres au SqlCommand
    sqlcmdTotal.Parameters.Add(resultat);

    //On exécute
    sqlcmdTotal.ExecuteNonQuery();

    total.Text = resultat.Value.ToString();
}
```

# ADO.NET, fonction avec paramètre en IN

```
private void nbJoueursEquipe() {  
    SqlCommand cmdTotal = new SqlCommand("totalJoueurEquipe", sqlconn);  
    cmdTotal.CommandText = "totalJoueurEquipe";  
    cmdTotal.CommandType = CommandType.StoredProcedure;  
  
    //Paramètre en IN  
    SqlParameter paramEquipe = new SqlParameter("@nomEquipe", SqlDbType.VarChar, 30);  
    paramEquipe.Direction = ParameterDirection.Input;  
    paramEquipe.Value = nomEquip; //'Canadiens'  
  
    //paramètre de retour  
    SqlParameter resultat = new SqlParameter("@totaEtudiant", SqlDbType.Int);  
    resultat.Direction = ParameterDirection.ReturnValue;  
  
    //ajouter les paramètres au SqlCommand  
    cmdTotal.Parameters.Add(resultat);  
    cmdTotal.Parameters.Add(paramEquipe);  
  
    cmdTotal.ExecuteNonQuery();  
    totalParEquipe.Text = resultat.Value.ToString();  
}
```

# ADO.NET

› Cas d'une fonction table:

Le cas d'une fonction table est traité comme une requête SQL SELECT quelconque.

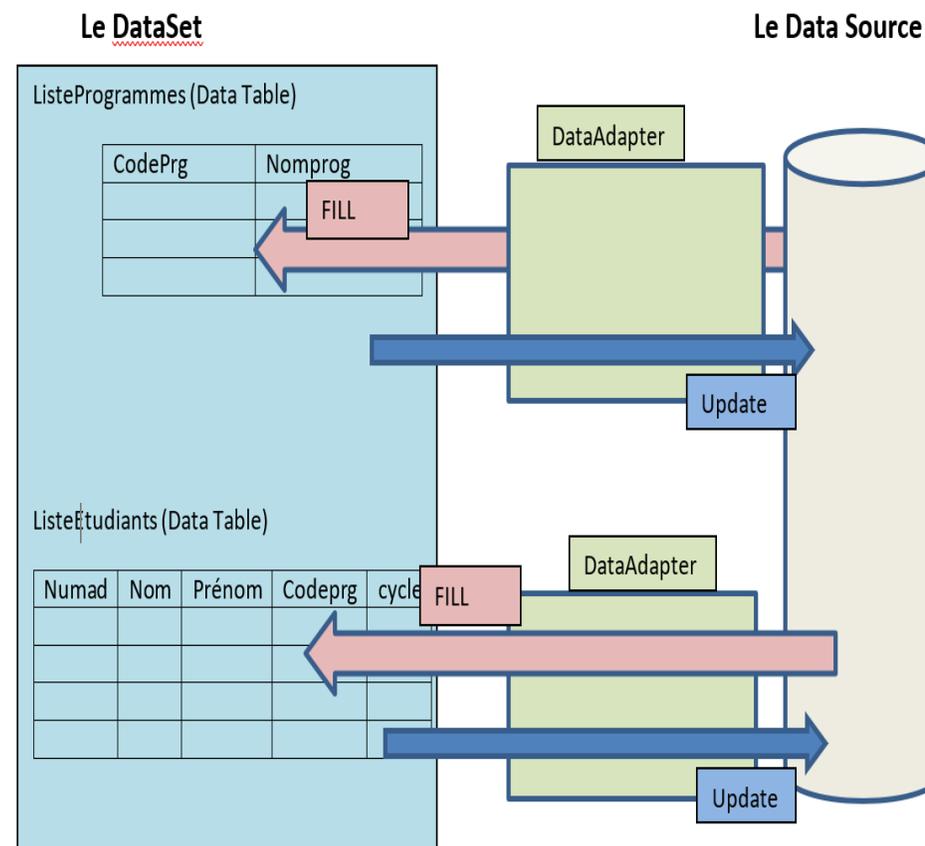
Pour exécuter une fonction table dans Management Studio on fait :  
`select * from nomfonction().`

# ADO.NET, définition

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.

L'interaction avec les sources de données existantes est contrôlée par le DataAdapter

Un ou plusieurs objets DataAdapter peuvent gérer la communication vers chaque source de données



# ADO.NET, le DataSet

Dans ce qui suit, nous allons présenter le DataSet très brièvement. Nous allons présenter:

- › Comment remplir un DataSet ? → avec l'objet SqlDataAdapter.
- › Comment afficher le contenu d'un DataSet ? → par un DataBinding.

L'avantage de présenter un DataSet est de vous aider au Tp2 pour l'affichage de données quelque soit le nombre de colonnes ramenées

# ADO.NET, le DataSet

## Propriétés importantes du DataSet:

Tables: Obtient la collection des tables contenues dans DataSet

## Méthodes importantes:

Clear(): Efface toutes les données du DataSet, en supprimant toutes les lignes et l'ensemble des tables.

AcceptChanges(): Valide toutes les modifications apportées à ce **DataSet** depuis son chargement ou depuis le dernier appel à **AcceptChanges**.

## Création du DataSet:

```
private DataSet monDataset = new DataSet();
```

# ADO.NET, SqlDataAdapter

L'objet SqlDataAdapter 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.

Constructeurs:

<b>SqlDataAdapter()</b>	<b>Instancie un objet SqlDataAdapter</b>
<b>SqlDataAdapter(String, SqlConnection)</b>	<b>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)</b>
<b>SqlDataAdapter(SqlCommand)</b>	<b>Initialise une nouvelle instance de la classe SqlDataAdapter avec l'instruction SQL SELECT spécifiée. (Ici, le SELECT est contenu dans le SqlCommand)</b>

# Le DataBinding

Le Data Bindings est l'action d'associer un Data Source à des contrôles comme des TextBox, des DataGridView ou encore des ListBox.

Plusieurs composants visuels (Form, TextBox, ListBox,...) possèdent une propriété DataSource et /DataBindings qui permet d'associer directement une source de données au composant en question. La source de données peut-être le résultat d'une requête SQL (select).

Lorsqu'il s'agit de lier un DataGridView à une source de données, il suffit d'utiliser la propriété DataSource du DataGridView (puis que le DGV a la même structure que le DataTable du DataSet)

```
BindingSource maSource = new BindingSource(monDataSet,  
"nomTableDansDataSet");
```

La ligne précédente indique que la variable maSource est liée à la table **nomTableDansDataSet** du DataSet monDataSet.

```
DGVEtudiants.DataSource = maSource;
```

La ligne précédente indique le DataGridView DGVEtudiants a comme source de données le contenu de la variable: maSource.

# Exemple

The screenshot shows a Windows form titled "Form1" with the following elements:

- Buttons: "connection" (highlighted with a blue border), "statut", and "Quitter".
- Text: "Liste des catégorie" above a dropdown menu.
- Text: "Selon l'énoncé" below a text input field.
- Text: "Toutes les questions" below a large grey rectangular area.

[Le code:](#)

Voir également le contenu du cours sur le site page 38 à 42 du <http://salihayacoub.com/420Kba/Theorie/2020ADO.pdf>



CONCLUSION



QUESTIONS ??