

A dark blue vertical bar on the left side of the page. A blue arrow-shaped banner points to the right from the bar, containing the text "Automne 2019".

Automne 2019

# ADO.NET, suite

Appels de procédures stockées et  
notion de DataSet.

Several thin, curved lines in shades of blue and grey originate from the bottom left corner and curve upwards and to the right.

Saliha Yacoub  
CLG

## Table des matières

Chapitre1, Rappels pour bien commencer.....	2
Connexion à la base de données : .....	2
Rappels des objets importants : .....	3
SqlCommand : .....	3
Quelques propriétés de l'objet SqlCommand .....	4
Méthodes importantes de l'objet SQLCommad.....	4
SqlDatareader .....	4
Quelques propriétés importantes de l'SqlDataReader .....	5
Quelques méthodes importantes de SqlDataReader.....	5
Chapitre 2, l'objet SqlParameter .....	6
Quelques Constructeurs.....	6
Propriétés importantes SqlParameter .....	6
Méthodes importantes de l'objet OrcaleParameter.....	7
Pour résumer : (pour l'instant).....	7
Exemple 1 : Insertion dans la base de données : TOUS les paramètres sont en IN .....	7
Exemple 2 : Appel de procédure stockée sans paramètres (ni en IN ni en OUT) .....	9
Exemple 3, Appel de procédures avec un paramètre en IN.....	10
Exemple 4, appel d'une fonction stockée qui retourne un scalaire.....	11
Exemple 5, appel d'une fonction stockée qui retourne une table.....	12
Chapitre 3, Le DataSet.....	13
Propriétés importantes de l'objet DataSet .....	14
Méthodes importantes de l'objet <b>DataSet</b> .....	14
L'objet SqlDataAdapter .....	15
Quelques constructeurs de l'objet SqlDataAdapter.....	15
Propriétés importantes de l'objet SqlDataAdapter.....	16
Méthodes importantes de l'objet SqlDataAdapter.....	16
Événements importants de l'objet SqlDataAdapter .....	16


## Chapitre1, Rappels pour bien commencer

Tous ce que vous avez appris sur ADO.NET dans le cours 420-KB6-LG, de la session 2 reste valable.

ADO. NET l'une des méthodes est utilisée pour accéder aux bases de données relationnelles (Oracle, MS SQL Server, MySQL, SQLite ....) côté application

Les classes utilisées par ADO.net selon le SGBD

<b>SGBDR : ORACLE</b>	<b>SGBDR : MS SQL Server</b>
OracleConnection	SqlConnection
OracleCommand	SqlCommand
OracleDataReader	SqlDataReader
OracleParameter	SqlParameter

Attention : 

L'ensemble des méthodes pour les classes MS SQL server sont les même que pour Oracle : (Open(), ExecuteNonQuery(),etc ...

De même l'ensemble des propriétés sont les mêmes : ConnectionString, State...

MS SQL Server, connexion à la base de données :

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

- Ce qu'il faut savoir:
  - Pas besoin de références supplémentaires.
  - L'espace de nom est System.Data.SqlClient;
  - Les classes sont : SqlConnection, SqlCommand etc.

Connexion à la base de données :

Avant d'établir une connexion à la base de données, de n'importe quel serveur, vous devez connaître le format de chaîne de connexion au serveur. Pour MS SQL server cette chaîne est de la forme :

"Data source =votreserveur; Intial Catalog = nomBd; User Id = user; Passsword =pass "


Exemple :

```
using System.Data.SqlClient;
namespace SqlAdo
{
    3 références
    public partial class Form1 : Form
    {
        1 référence
        public Form1()
        {
            InitializeComponent();
        }
        SqlConnection conn = new SqlConnection();
        1 référence
        private void btnConnecter_Click(object sender, EventArgs e)
        {
            try
            {
                string dSource = "Data Source=M-INFO-SY\\SQLEXPRESS2017;Initial Catalog=BdJeu;User ID=patoche;Password=123456789";
                conn.ConnectionString = dSource;
                conn.Open();
                MessageBox.Show(conn.State.ToString());
            }
            catch(Exception ex)
            { MessageBox.Show(ex.Message); }
        }
    }
}
```

Rappels des objets importants :

SqlCommand :

L'objet SqlCommand contient les commandes envoyées aux SGBD. Ces commandes sont envoyées soit en utilisant des requêtes simples, soit en utilisant des procédures stockées. Lorsque la requête SQL ou procédure retourne un résultat, il est retourné dans un OracleDataReader

Attention : 

Pour envoyer une requête à la base de donnée, on utilise un objet SqlCommand, en fournissant une connexion et une requête SQL. Dans les cas qui suivent la commande SQL représente le **nom d'une procédure stockée**

```
SqlCommand objCommand = new SqlCommand("nomProcedure", nomConnection);
```

### Quelques propriétés de l'objet SqlCommand

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 <b>CommandText</b> doit être interprétée (instruction SQL ou <b>procédure</b> )
Connection	Obtient ou définit l'objet <b>OracleConnection</b> utilisé par cette instance de <b>OracleCommand</b> .
Parameters	Spécifie les paramètres de la requête SQL ou <u>de la procédure stockée</u>


### Méthodes importantes de l'objet SQLCommad

ExecuteNonQuery()	Exécute une instruction SQL sur <b>Connection</b> et retourne le nombre de lignes affectées.
ExecuteReader()	Surchargé. Envoie <b>CommandText</b> à <b>Connection</b> et génère <b>OracleDataReader</b>
ExecuteScalar()	Exécute la requête et retourne la première colonne de la première ligne.

### SqlDatareader

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

L'objet SqlDataReader se crée par la méthode ExecuteReader de l'objet SqlCommand

Attention : 

**L'objet SqlDataReader NE SE Crée pas ave la méthode new .**

**Tout ce que vous avez vu sur OracleDataReader ne change pas pour SqlDataReader même si nous sommes en procédures stockées, car son but est de récupérer un ensemble de résultats.**

## Quelques propriétés importantes de SqlDataReader

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

## Quelques méthodes importantes de SqlDataReader

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

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

Nouveau :

## Chapitre 2, l'objet SqlParameter

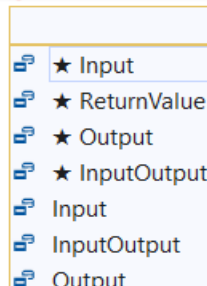
L'objet SqlParameter représente un paramètre pour l'SqlCommand ou une colonne du DataSet

### Quelques Constructeurs

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

La direction des paramètres peut être :

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



Input, lorsque les paramètres de la procédure sont en IN

OutPut lorsque les paramètres de la procédure sont en OUT

ReturnValue lorsqu'il s'agit d'une fonction

### Propriétés importantes SqlParameter

Direction	Obtient ou définit une valeur qui indique si le paramètre est un paramètre d'entrée uniquement, de sortie uniquement, bidirectionnel ou de valeur de retour d'une procédure stockée.
ParameterName	Obtient ou définit le nom de SqlParameter
SqlDbType	Spécifie le type de données Sql
Size	Obtient ou définit la taille maximale, en octets, des données figurant dans la colonne.
Value	Obtient ou définit la valeur du paramètre

## Méthodes importantes de l'objet OracleParameter

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

Pour résumer : (pour l'instant)

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

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

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

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

Voici le code de la procédure InsertQuestion

```
create procedure InsertQuestion
(@enonce varchar(200), @flag int,@difficulte varchar(20),@categ int) as
begin
insert into Questions values(@enonce,@flag,@difficulte,@categ);
end;
```



Voici le code C# qui appelle la procédure :

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

```
private void inserer()
{
    try
    {
        //on prépare la requête, qui dans notre cas une procédure stockée
        //remarque : la chaîne dans SqlCommand indique le nom de la procédure
        //remarque : CommandText a le nom de la procédure
        //remarque : CommandType dit que c'est une procédure.

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

        //declaration des paramètres

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

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

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

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

        //affectation des valeurs aux paramètres

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

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

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

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

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

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

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

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

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

### Exemple 3, Appel de procédures avec un paramètre en IN

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

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

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

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

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

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

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

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

La fonction suivante retourne le nombre total de Question

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

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

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

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

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

    }

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

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

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

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

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

### Code ADO.NET

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

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

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

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

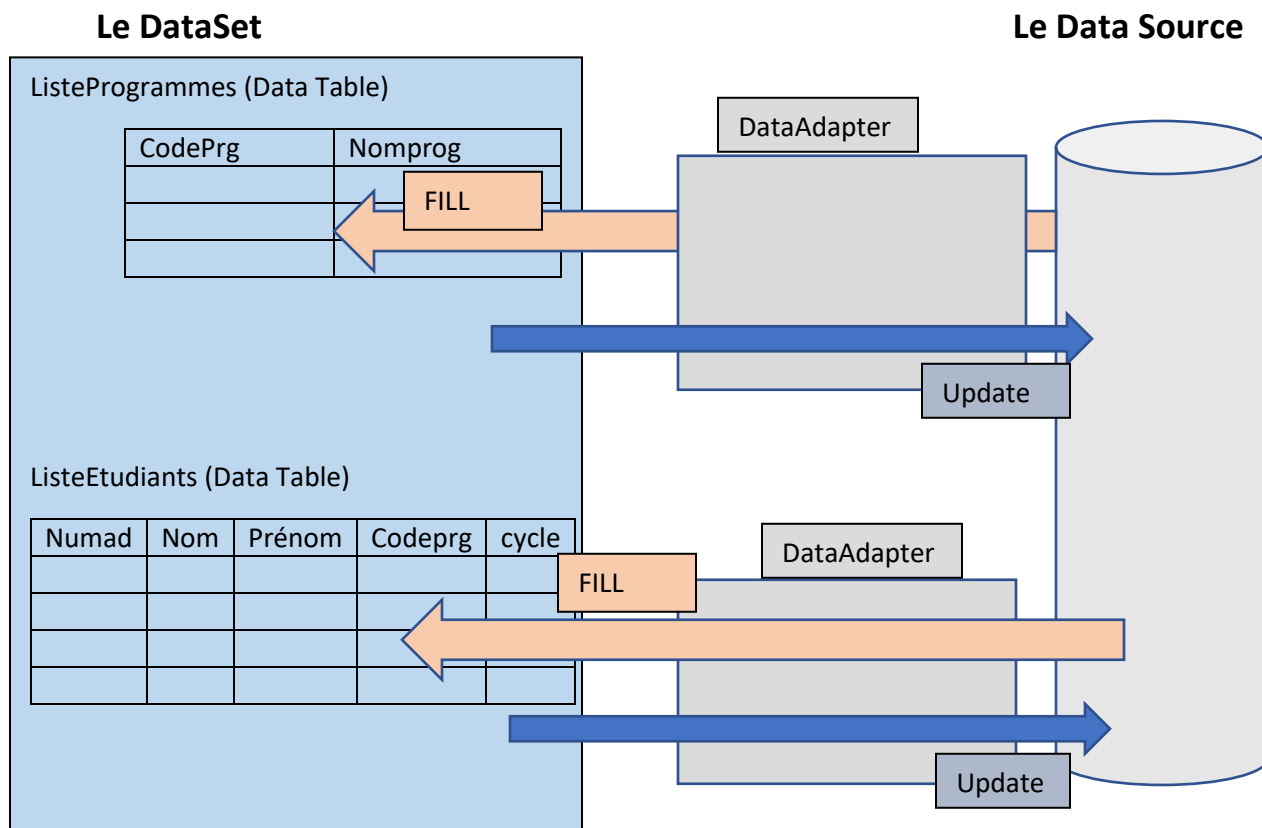
        while (resQuestion.Read())
        {
            dgvQuestions.Rows.Add(resQuestion.GetString(0), resQuestion.GetString(1));
        }
        resQuestion.Close();
    }

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

## Chapitre 3, Le DataSet

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

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



## Propriétés importantes de l'objet DataSet

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

## Méthodes importantes de l'objet DataSet

<b>AcceptChanges()</b>	Valide toutes les modifications apportées à ce <b>DataSet</b> depuis son chargement ou depuis le dernier appel à <b>AcceptChanges</b> .
<b>Clear()</b>	Efface toutes les données de <b>DataSet</b> en supprimant toutes les lignes de l'ensemble des tables.
<b>Clone()</b>	Copie la structure de <b>DataSet</b> , y compris tous les schémas, relations et contraintes <b>DataTable</b> . Ne copie aucune donnée.
<b>Copy()</b>	Copie à la fois la structure et les données de ce <b>DataSet</b> .
<b>GetChanges()</b>	Obtient une copie du <b>DataSet</b> contenant l'ensemble des modifications qui lui ont été apportées depuis son dernier chargement ou depuis l'appel à <b>AcceptChanges</b> .
<b>GetXml()</b>	Retourne la représentation XML des données stockées dans <b>DataSet</b> .
<b>GetXmlSchema()</b>	Retourne le schéma XSD de la représentation XML des données stockées dans <b>DataSet</b> .

<b>HasChanges</b>	Obtient une valeur indiquant si <b>DataSet</b> contient des modifications, notamment des lignes nouvelles, supprimées ou modifiées.
<b>Merge</b>	Fusionne un <b>DataSet</b> , un <b>DataTable</b> ou un tableau d'objets <b>DataRow</b> dans le <b>DataSet</b> ou le <b>DataTable</b> en cours.
<b>RejectChanges</b>	Annule toutes les modifications apportées à <b>DataSet</b> depuis sa création ou le dernier appel à <b>DataSet.AcceptChanges</b> .
<b>Reset</b>	Rétablit l'état d'origine de <b>DataSet</b> . Les sous-classes doivent substituer <b>Reset</b> pour rétablir l'état d'origine de <b>DataSet</b> .

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

## L'objet SqlDataAdapter

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

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

## Quelques constructeurs de l'objet SqlDataAdapter

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



### Propriétés importantes de l'objet `SqlDataAdapter`

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

### Méthodes importantes de l'objet `SqlDataAdapter`

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

### Événements importants de l'objet `SqlDataAdapter`

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