Département d'informatique

420 –KED-LG, conception de bases de données

Introduction à ADO.NET

Préparé par Saliha Yacoub

Table des matières

1. Introduction

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

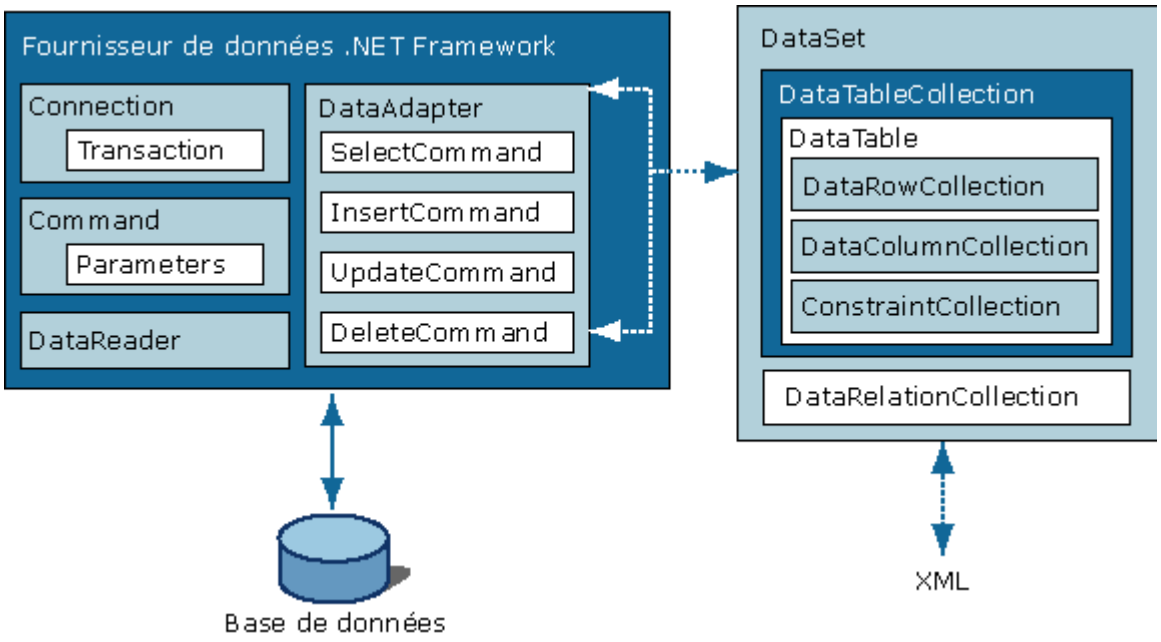
- Le modèle « **connecté** » qui est bien adapté aux applications à deux couches traditionnelles;
- Le modèle « **déconnecté** » qui est destinés aux applications à multicouches.

Les sources de données peuvent être :

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

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

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



Le fournisseur de données

Un fournisseur de données est utilisé pour :

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

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

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

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

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

Dans le document qui suit, et puisque notre base de données est ORACLE, nous utiliserons le provider ODP.NET, qui est le fournisseur d'accès aux données oracle

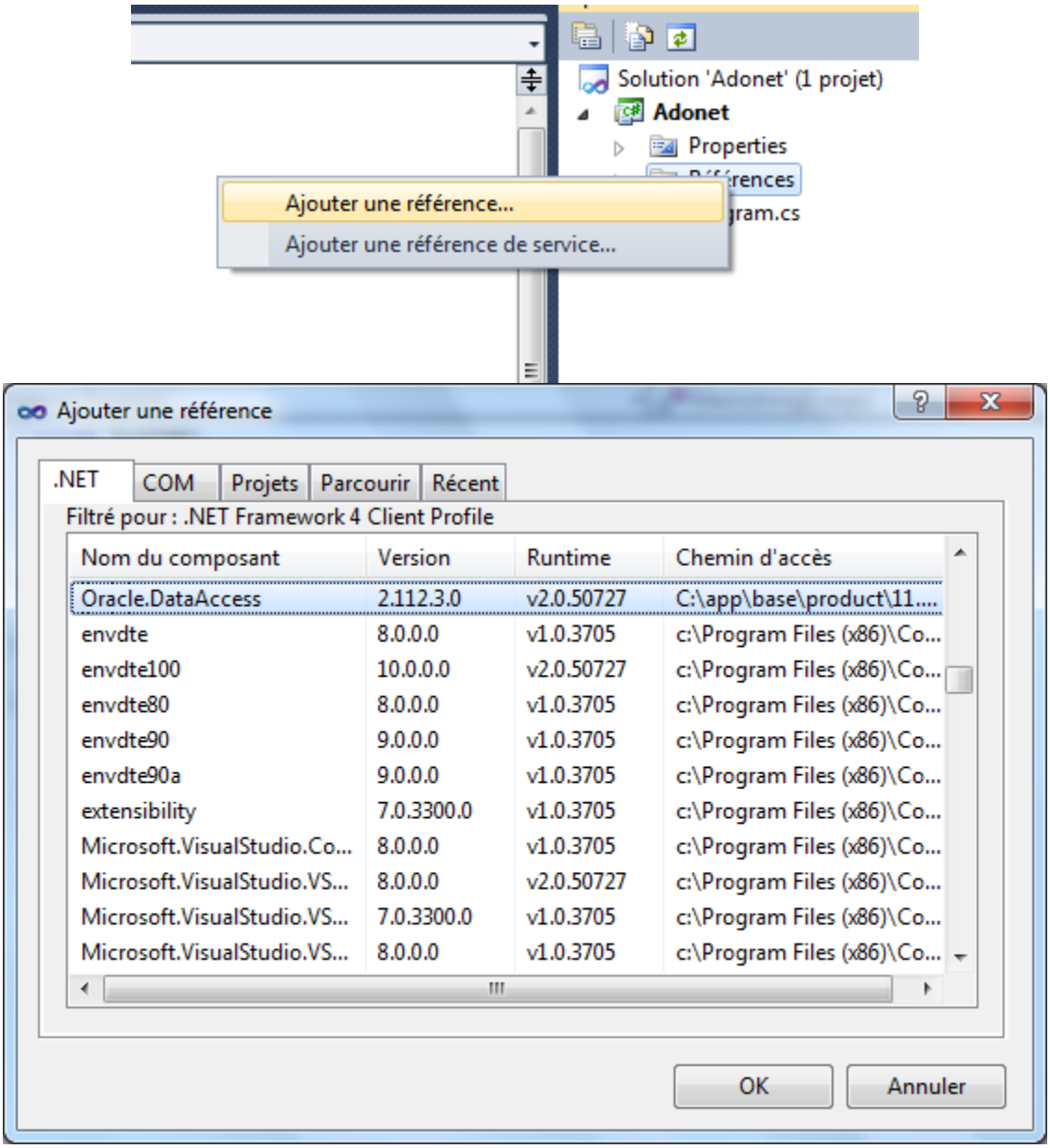
Un exemple utilisant OLEDB (avec Oracle et ACCESS) sera donné à la fin du document.

2. Prés-requis :

Pour pouvoir utiliser les classes d'ODP.net, il faudra :

1. Installer ODAC (Oracle Data Access Component) : pour la procédure d'installation allez sur http://docs.oracle.com/cd/B28359_01/appdev.111/b28844.pdf

2. Ajout de référence dans Visual studio. (C#).La référence est: **Oracle.DataAccess.dll**

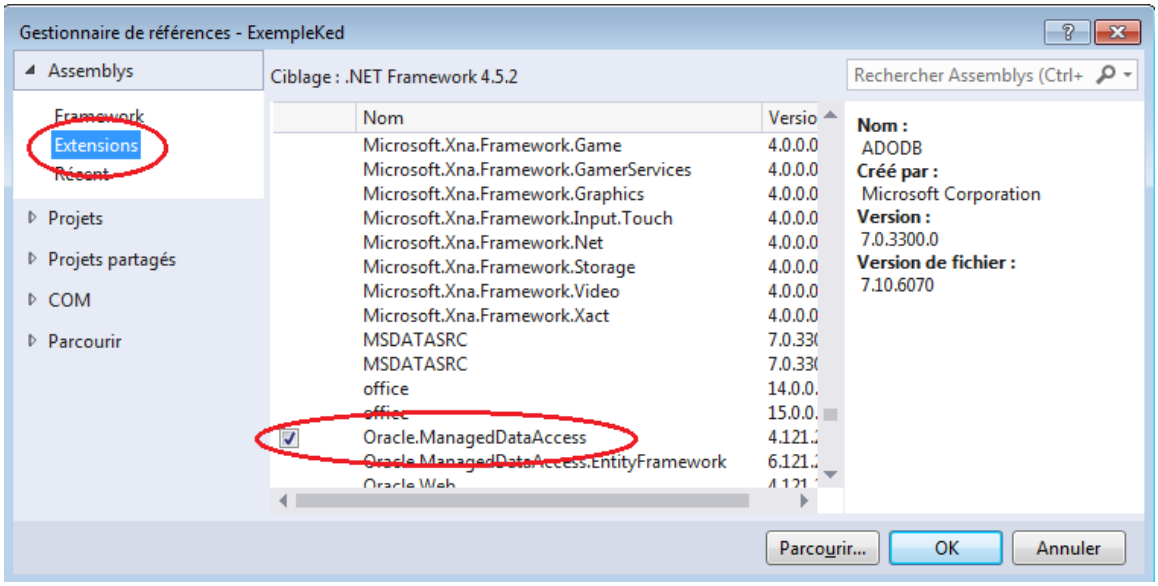


3. Ajout de l'espace de nom spécifique à ODP.net:
   using Oracle.DataAccess.Client;

```
Program.cs*
Adonet.Program                                  Main(strin
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Oracle.DataAccess.Client;

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

Important : Si au lieu d'utiliser ODAC, vous utilisez Oracle Developer Tools for Visual Studio 2015 (ce qui est le cas au CEGEP) Alors

La référence qui sera utilisée est :



Et l'espace de nom est :

```
using Oracle.ManagedDataAccess.Client;
```

3. L'objet OracleConnection

Description de l'objet OracleConnection

Un objet OracleConnection représente une connexion à la base de données Oracle. 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 **OracleConnection**. Les commandes circulent sur les connexions et des jeux de résultats sont retournés sous la forme de flux qui peuvent être lus par un objet **OracleDataReader**, ou placés dans un objet **DataSet** à l'aide de la méthode **FILL** de l'objet **OracleDataAdapter**

Propriétés importantes de l'objet OracleConnection

Propriété ConnectionString

Il s'agit de la chaîne de connexion qui comprend des paramètres requis pour établir la connexion initiale. La valeur par défaut de cette propriété est une chaîne vide ("").

Les principaux paramètres pouvant être inclus dans la chaîne sont les suivants :

- **Data source** : le nom de serveur ou de la source de données ;
- **User Id** : Compte de connexion Oracle ;
- **Password** : Mot de passe de la session du compte Oracle

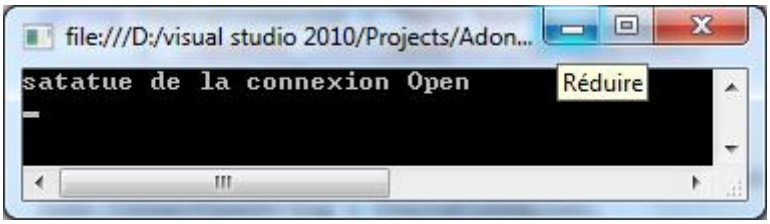
Le Data Source correspond à la description de la base de donnée, cette description est contenue dans le fichier tsnames.ora (situé dans le C:\app\product\11.2.0\client_1\Network\Admin). La description du Data Source est de la forme suivante :

```
OraDb=
  (DESCRIPTION=
    (ADDRESS_LIST=

(ADDRESS=(PROTOCOL=TCP)(HOST=ORASRVR)(POR
T=1521))
    )
    (CONNECT_DATA=
      (SERVER=DEDICATED)
      (SERVICE_NAME=ORCL)
    )
  )
```

Exemple: déclaration de la chaine de connexion de nom ChaineConnexion.

```
string ChaineConnexion = "Data Source=(DESCRIPTION="

+ "(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)"

+"(HOST=222.333.444.555)(PORT=1521)))"

+ "(CONNECT_DATA=(SERVICE_NAME=ORCL.clg.qc.ca)));"
+ "User Id=usager1;Password=passe1";
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Oracle.DataAccess.Client;

namespace Adonet
{
    class Program
    {
        static void Main(string[] args)
        {
            string ChaineConnexion = "Data Source=(DESCRIPTION="
                   + "(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)"
                   + "(HOST=222.333.444.555)(PORT=1521)))"
                   + "(CONNECT_DATA=(SERVICE_NAME=ORCL.clg.qc.ca)));"
                   + "User Id=user;Password=password";
            OracleConnection conn = new OracleConnection(ChaineConnexion);
            conn.ConnectionString = ChaineConnexion;
            try{
            // ouverture de la connexion
            conn.Open();

            System.Console.WriteLine("statut de la connexion" + " "+ conn.State);
            System.Console.Read();
            }
        catch (Exception ex)
            {
                System.Console.WriteLine(ex.Message);
            }
       // fermeture  de la connexion
        conn.Close();
        }
    }
}
```



Important :

Si vous ne voulez pas trainer dans votre code cette longue chaine de connexion, il suffit de donner un nom à votre base de données.

Pour cela, il suffit de trouver le fichier **tnsnames.ora** et donner un alias à votre BD. Pour cela, il suffit d'enlever le signe de commentaires (#) aux endroits indiqués. Évidemment, remplacer l'adresse IP 222.333.444.555 par l'adresse IP ou le nom de votre serveur de BD

```
primogene =
 (DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = 222.333.444.555)(PORT = 1521))
 (CONNECT_DATA =
#    (SERVER = DEDICATED)
   (SERVICE_NAME = orcl.clg.qc.ca)
  )
 )
```

Votre chaine de connexion sera alors

```
string chaine = "Data Source = primogene; User Id = user1; password = user1";
```

Le fichier tsnames.ora est dans **C:\app\product\11.2.0\client_1\Network\Admin**
Ou dans
C:\Program Files (x86)\Oracle Developer Tools for VS2015\Network\Admin

Quelques propriétés de l'objet OracleConnection

ConnectionString	Indique la chaine de connexion pour se connecter à la base de données Oracle
DataSource	Indique le nom de la source de données dans le fichier tsname.ora
State	Indique l'état actuel de de la connexion. ODP a deux valeurs pour cette propriété. Closed ou Open. Par défaut la valeur est Closed

Méthodes importantes de l'objet OracleConnection (Public)

CreateCommand	Crée et retourne un objet **OracleCommand** associé à **OracleConnection**.
Open	Ouvre une connexion à une base de données avec les paramètres de propriété spécifiés par **ConnectionString**.
Close	Ferme la connexion à la base de données. Il s'agit de la méthode privilégiée pour la fermeture de toute connexion ouverte

4. L'objet OracleCommand

L'objet OracleCommand 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 ou autre (voir **plus loin).**

a) Quelques propriétés de l'objet OracleCommand

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

b) Méthodes importantes de l'objet OracleCommad

ExecuteNonQuery	Exécute une instruction SQL sur **Connection** et retourne le nombre de lignes affectées.
ExecuteReader	Surchargé. Envoie **CommandText** à **Connection** et génère **OracleDataReader**
ExecuteScalar	Exécute la requête et retourne la première colonne de la première ligne.
Clone	Crée une copie de l'objet OracleCommande

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Oracle.DataAccess.Client;
using Oracle.DataAccess.Types;

namespace Adonet
{
    class Program
    {

        static void Main(string[] args)
        {

           string sql1 = "update etudiants set cycle = 2";

            string ChaineConnexion = "Data Source=(DESCRIPTION="
                   + "(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)"
                   + "(HOST=222.333.444.5515(PORT=1521)))"
                   + "(CONNECT_DATA=(SERVICE_NAME=ORCL.clg.qc.ca)));"
                   + "User Id=user1;Password=password";
            OracleConnection conn = new OracleConnection(ChaineConnexion);


             //ouverture de la connection
            conn.Open();

            System.Console.WriteLine("satatue de la connexion" + " "+ conn.State);
            System.Console.Read();

            OracleCommand objCommand = new OracleCommand(sql1, conn);
            int res = objCommand.ExecuteNonQuery();
            System.Console.WriteLine("nombre de lignes mises a jour" + " " + res);


            System.Console.Read();

       //fermeture de la connection
            conn.Close();

        }
    }
}
```

5. L'objet OracleDataReader

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 OracleDataReader se crée par la méthode **ExecuteReader** de l'objet **OracleCommand.**

a) Quelques propriétés importantes de l'OracleDataReader

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

b) Quelques méthodes importantes de L'OracleDataReader

Colse	Ferme l'objet OracleDataReader
Dispose	Libère toutes les ressources occupée par l'objet OracleDataReader
Read	Permet d'avancer l'objet OracleDataReader jusqu'à l'enregistrement suivant.
GetDateTime	Obtient la valeur de la colonne spécifiée sous la forme d'un objet DateTime.
GetDecimal	Obtient la valeur de la colonne spécifiée sous la forme d'un objet Decimal.
GetDouble	Obtient la valeur de la colonne spécifiée sous la forme d'un nombre de type Double.
GetFloat	Obtient la valeur de la colonne spécifiée sous la forme d'un nombre de type Float.
GetInt16	Obtient la valeur de la colonne spécifiée sous la forme d'un entier signé 16 bits.
GetInt32	Obtient la valeur de la colonne spécifiée sous la forme d'un entier signé 32 bits.
GetInt64	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	Obtient le nom de la colonne spécifiée.
GetString	Obtient la valeur de la colonne spécifiée sous la forme d'une chaîne.

Consulter http://docs.oracle.com/html/B28089_01/OracleDataReaderClass.htm#i1004157 pour le reste des méthodes.

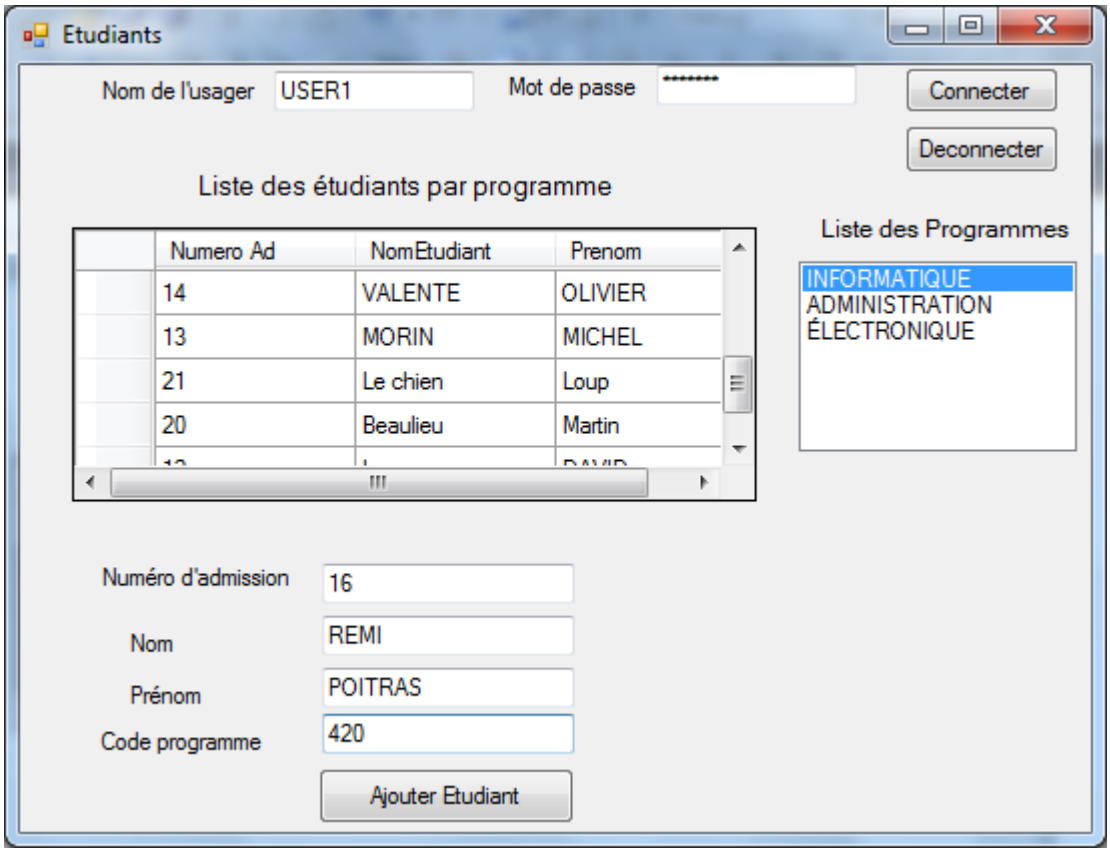
Par défaut, un **DataReader** charge une ligne entière en mémoire à chaque appel de la méthode **Read**. Il est possible d'accéder aux valeurs de colonnes soit par leurs noms soit par leurs références ordinales. Une solution plus performante est proposée permettant d'accéder aux

valeurs dans leurs types de données natifs (**GetInt32**, **GetDouble**, **GetString**). Par exemple si la première colonne de la ligne indicée par 0 est de type **int**, alors il est possible de la récupérer à l'aide de la méthode **GetInt32** de l'objet **DataReader**.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Oracle.DataAccess.Client;
using Oracle.DataAccess.Types;

namespace Adonet
{
    class Program
    {

        static void Main(string[] args)
        {

   string sql1 = "update etudiants set cycle = 2";
   string sql2 = "SELECT nom, prenom, codeprg  FROM ETUDIANTS where codeprg =420";

            string ChaineConnexion = "Data Source=(DESCRIPTION="
                   + "(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)"
                   + "(HOST=222.333.444.555)(PORT=1521)))"
                   + "(CONNECT_DATA=(SERVICE_NAME=ORCL.clg.qc.ca)));"
                   + "User Id=user;Password=password";
            OracleConnection conn = new OracleConnection(ChaineConnexion);
             //ouverture de la connection
            conn.Open();

            System.Console.WriteLine("satatue de la connexion" + " "+ conn.State);
            System.Console.Read();
            OracleCommand objCommand = new OracleCommand(sql1, conn);
            int res = objCommand.ExecuteNonQuery();
            System.Console.WriteLine("nombre de lignes mises a jour" + " " + res);
            System.Console.ReadLine();

        OracleCommand ObjSelct = new OracleCommand(sql2, conn);
        OracleDataReader ObjeRead = ObjSelct.ExecuteReader();
            while (ObjeRead.Read())
            {
              Console.WriteLine("{0} - {1} - {2}",
              ObjeRead.GetString(0), ObjeRead.GetString(1), ObjeRead.GetInt32(2));
            }
            ObjeRead.Close()

       System.Console.Read();
            conn.Close();
        }
    }
}
```

6. Exemple C# Windows Forms

Dans l'exemple qui suit, après s'être connecté, nous voulons avoir le résultat suivant.



Deux zones de textes avec un bouton **connecter** permettent d'obtenir la connexion à la base de données.

Un bouton déconnecter qui permet de fermer la connexion et l'application.

Attention, dans ce code, certains try {}..catch {} ne sont pas là. Il faudra faire les validations

Nous avons une LisBox de nom **`ListeProgrammes`** qui va contenir la liste des programmes disponible dans la base de données. Cette liste est remplie dès la connexion à la BD

La requête permettant de remplir cette liste est sql3 et la fonction correspondante est

```
private void ListerProgramme()
```

En fonction de du programme qui est sélectionné dans la liste, la liste des étudiants s'affiche dans le DataGridView, (voir les figures qui suivent) dont le nom est DGVEtudiants. La requête permettant de remplir de DGVEtudiant est sql2 et la fonction est

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

Dans la fonction **ListerProgramme(),** nous avons utilisez un objer OracleCommand de nom **OraCmdProg** pour envoyer la requête SQL à la base de données.

Pour récupérer les résultat du SELECT, nous avons utilisez la méthode **ExecuteReader()** de l'objet **OraCmdProg** pour récupérer les résultats dans un objet **OracleDatareader** de nom **objRead**

Pour pouvoir accéder à tous les enregistrements contenu l'objet **objRead**, nous avons utilisé une boucle while avec la méthode **Read()** de l'objet **OracleDatareader objRead.** Rappelons que la méthode **Read()** permet d'avancer l'objet OracleDataReader jusqu'à l'enregistrement suivant et que par défaut, un **DataReader** charge une ligne entière en mémoire à chaque appel de la méthode **Read**

Par la suite, pour d'accéder aux valeurs de colonnes de l'objet **objRead,** nous y accédons aux valeurs dans leurs types de données natifs. Dans notre cas la requête est

string sql3 = "select nomprog from programmes" et elle retourne un varchar2 dans Oracle. On utilise alors la méthode **GetString(0)** de l'objet **objRead.** Le chiffre à l'intérieur de la parenthèse de la méthode (paramètre de la méthode) indique l'indice de colonne dans l'objet **OracleDatareader.** La première colonne a toujours l'indice **0(zéro).**

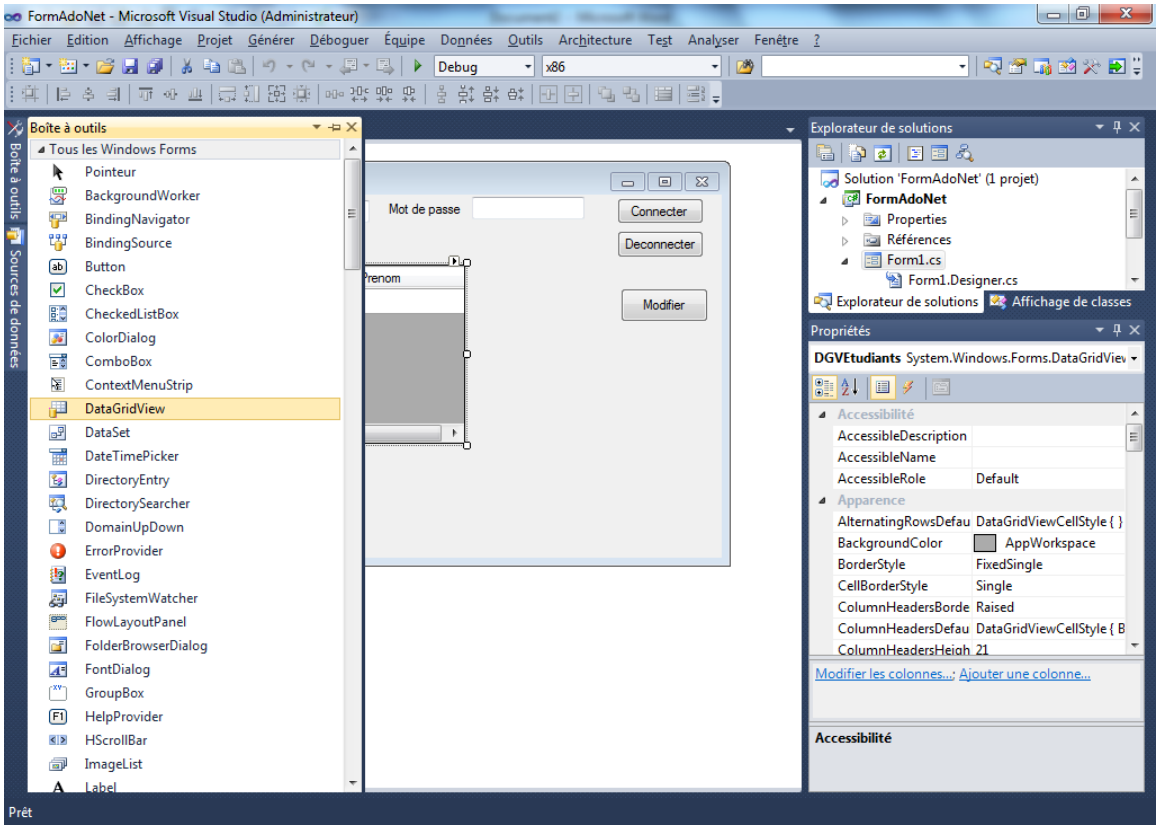
Le même principe est utilisé dans la fonction :

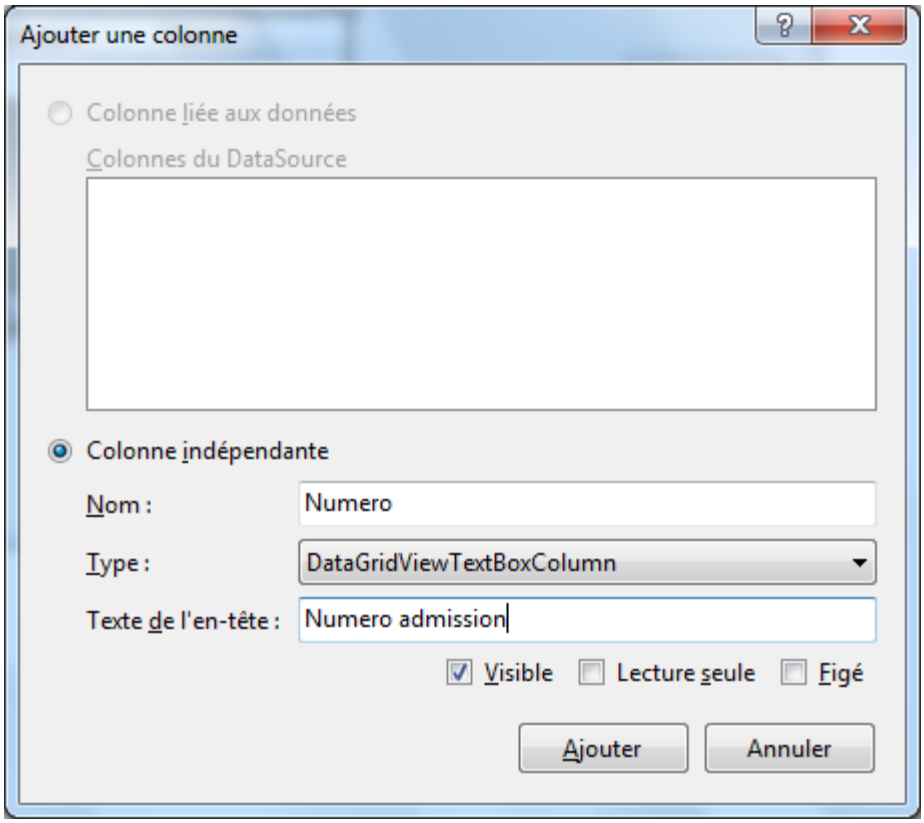
```
ListeProgrammes_SelectedIndexChanged(object sender, EventArgs e)
```

Un objet OracleCommand **oraCmd** pour passer la commande SQL
La méthode **ExecuteReader();** sur l'objet oraCmd permet d'avoir les résultats de la requêtre dans un OracleDataReader **oraRead**

La requête étant SELECT NUMAD, NOM, PRENOM FROM …(VOIR PLUS BAS), l'objet **oraRead** contient un ensemble d'enregistrement de trois colonnes (NUMAD, de type number et NOM, PRENOM de type varchar2). La méthode Read() et une boucle **while** est utilisée pour lire les enregistrement de **oraRead.** Les méthodes utilisées pour accéder aux valeurs des colonnes sont :
GetInt32(0)→pour NUMAD
GetString(1)→pour nom
GetString(2)→pour prénom.

Le résultat est ensuite envoyé dans DataGridView DGVEtudiants en utilisant la propriété **Rows** et la méthode **Add()** de l'objet DGVEtudiants


FormAdoNet - Microsoft Visual Studio (Administrateur)
Fichier Edition Affichage Projet Générer Déboguer Équipe Données Outils Architecture Test Analyser Fenêtre ?
Debug
x86
Boîte à outils
Tous les Windows Forms
Pointeur
BackgroundWorker
BindingNavigator
BindingSource
Button
CheckBox
CheckedListBox
ColorDialog
ComboBox
ContextMenuStrip
DataGridView
DataSet
DateTimePicker
DirectoryEntry
DirectorySearcher
DomainUpDown
ErrorProvider
EventLog
FileSystemWatcher
FlowLayoutPanel
FolderBrowserDialog
FontDialog
GroupBox
HelpProvider
HScrollBar
ImageList
Label
Mot de passe
Connecter
Deconnecter
Modifier
Explorateur de solutions
Solution 'FormAdoNet' (1 projet)
FormAdoNet
Properties
Références
Form1.cs
Form1.Designer.cs
Affichage de classes
Propriétés
DGVEtudiants System.Windows.Forms.DataGridVie
Accessibilité
AccessibleDescription
AccessibleName
AccessibleRole Default
Apparence
AlternatingRowsDefau DataGridViewCellStyle { }
BackgroundColor AppWorkspace
BorderStyle FixedSingle
CellBorderStyle Single
ColumnHeadersBorde Raised
ColumnHeadersDefau DataGridViewCellStyle { B
ColumnHeadersHeigh 21
Modifier les colonnes...; Ajouter une colonne...
Accessibilité
Prêt


Ajouter une colonne
Colonne liée aux données
Colonnes du DataSource
Colonne indépendante
Nom :
Numero
Type :
DataGridViewTextBoxColumn
Texte de l'en-tête :
Numero admission
Visible
Lecture seule
Figé
Ajouter
Annuler

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Oracle.DataAccess.Client;

namespace FormAdoNet
{
    public partial class GestEtudiants : Form
    {
        public GestEtudiants()
        {
            InitializeComponent();
        }

        private OracleConnection conn = new OracleConnection();
        string sql3 = "select nomprog from programmes";
        //string sql4 = "select codeprg from programmes";

        private void Form1_Load(object sender, EventArgs e)
        {

        }
        private void Connect_Click(object sender, EventArgs e)
        {
            try
            {
                string Dsource = "(DESCRIPTION="
                + "(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)"
                + "(HOST=205.237.244.251)(PORT=1521)))"
                + "(CONNECT_DATA=(SERVICE_NAME=ORCL.clg.qc.ca)))";

                String ChaineConnexion = "Data Source=" + Dsource
                   + ";User Id=" + textUser.Text + ";Password=" + textPasswd.Text;
                conn.ConnectionString = ChaineConnexion;

                conn.Open();

                MessageBox.Show(conn.State.ToString());

      //lister tous les programmes dès la connexion
                ListerProgramme();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message.ToString());
            }

        }
```

```
        private void deconnect_Click(object sender, EventArgs e)
        {
            conn.Close();
            Application.Exit();
        }

        // lister tous les programmes  et les mettre dans la ListBox.
        // le nom de cette LisBox est ListeProgrammes.
        private void ListerProgramme()
        {
            try
            {
                OracleCommand oraCmdProg = new OracleCommand(sql3, conn);
                oraCmdProg.CommandType = CommandType.Text;

                OracleDataReader objRead = oraCmdProg.ExecuteReader();

                while (objRead.Read())
                {
                    //ListeProgrammes.Items.Add(objRead.GetInt32(0));
                    ListeProgrammes.Items.Add(objRead.GetString(0));
                }

                ListeProgrammes.SelectedIndex = 0;
                objRead.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message.ToString());
            }
        }

      //lorqu'on change le contenu selectionné de la ListBox, on obtient la liste
des étudiants qui seron affichés dans un DataGridView. Ajouter les try ..catch
     private void ListeProgrammes_SelectedIndexChanged(object sender, EventArgs e)
        {
            DGVEtudiants.Rows.Clear();

            string sql2 = "SELECT NUMAD, NOM, PRENOM FROM ETUDIANTS E " +
            "INNER JOIN PROGRAMMES P ON e.codeprg = p.codeprg where nomprog =" +
            "'" + ListeProgrammes.Text + "'";

            OracleCommand oraCmd = new OracleCommand(sql2, conn);
            oraCmd.CommandType = CommandType.Text;
            OracleDataReader oraRead = oraCmd.ExecuteReader();
            while (oraRead.Read())
            {
              DGVEtudiants.Rows.Add(oraRead.GetInt32(0),
              oraRead.GetString(1), oraRead.GetString(2));
                // DGVEtudiants.Rows.Add(oraRead[0], oraRead[1], oraRead[2]);

            }
            oraRead.Close();
        }

    }
}
```

Dans la figure de la page 12, nous avons également une fonction qui permet d'ajouter un enregistrement à la table « étudiants» par le bouton **Ajouter Etudiant** de nom **Insertion**

Le code permettant de faire ces insertions est le suivant :

```
private void Insertion_Click(object sender, EventArgs e)
        {
            string sqlIns = "insert into etudiants(numad, nom, prenom,codeprg)
values ("+ textNumad.Text + ",'" + textNom.Text + "'," + "'" + textPrenom.Text +
"'," + textCodePrg.Text + ")";

            try
            {
                OracleCommand orainsert = new OracleCommand(sqlIns, conn);
                orainsert.CommandType = CommandType.Text;
                orainsert.ExecuteNonQuery();

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

Un objet OracleCommand **orainsert** permet d'envoyer la requête SQL INSERT INTO.

Les valeurs des colonnes à insérer sont passées par les textBox correspondant.

La méthode **ExecuteNonQuery()** de l'objet **orainsert** permet d'exécuter la requête.

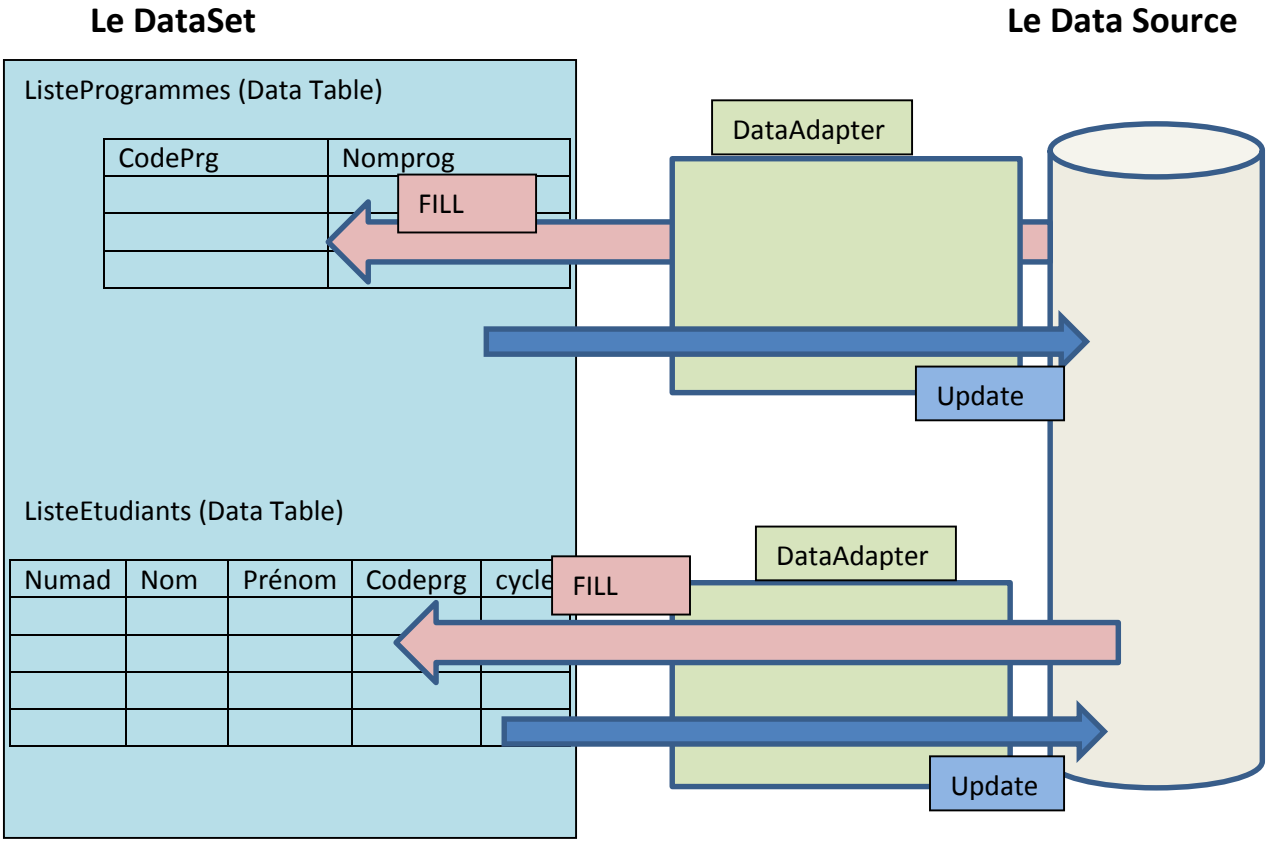
Une fonction vider() permettant de vider les zones de textes après insertion a été ajoutée, pour permettre d'effacer les zones de texte et de faire une autre insertion. Le code est le suivant :

```
private void vider()
        {
            textNumad.Clear();
            textNom.Clear();
            textPrenom.Clear();
            textCodePrg.Clear();
        }
```

7. 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`

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

Méthodes importantes de l'objet DataSet

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

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

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

8. L'objet OrcaleDatatAdapter

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 OracleDataAdpter

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

Propriétés importantes de l'objet OracleDataAdpter

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

Méthodes importantes de l'objet OracleDataAdpter

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

Événements importants de l'objet OracleDataAdpter

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

9. L'objet OracleParameter

L'objet OrcaleParameter représente un paramètre pour l'OracleCommand ou une colonne du DataSet

Quelques Constructeurs

OracleParameter()	Instancie un OrcaleParameter
OracleParameter (string, OracleDbType)	String désigne le nom du paramètre, le OracleDbtype désigne le type de données du Pramètre (un OracleType.) : public OracleParameter(string parameterName, OracleDbType oraType)
OracleParameter(string, OracleDbType, int)	Même que le précédent, sauf qu'on indique la taille du paramètre
OracleParameter(string, OracleDbType, int, string)	Même que le précédent, sauf qu'on indique la taille du paramètre et le nom de la colonne source
OracleParameter(string, OracleDbType, ParameterDirection)	Le ParameterDirection indique si le paramètre est en IN ou Out. Utilisé lorsque nous avons une procédure stockée

http://docs.oracle.com/html/E15167_01/OracleParameterClass.htm#i1011127

Propriétés importantes OrcaleParameter

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 OracleParameter
OracleDbType	Spécifie le type de données Oracle
Size	Obtient ou définit la taille maximale, en octets, des données figurant dans la colonne.
Value	Obtient ou définit la valeur du paramètre

Méthodes importantes de l'objet OrcaleParameter

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

10. Le Data Binding (liaison de données)

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

Le fait de lier les contrôle à la source de données va éviter de parcourir la source est de lire les données une à une afin de les afficher.

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).

Liaison avec une zone de Texte : Des données provenant d'une source de données peuvent-être affichées dans de TextBox, à condition de prévoir un moyen de navigation à l'intérieur de la source de données (bouton «next» et «previous»)

La liaison d'un DataSource à un TextBox se fait par sa propriété DataBindings

```
textNumad.DataBindings.Add("Text", monDataSet, "ListeEtudiants.Numad")
```

La méthode Add appliquée au DataBindings prend trois arguments :

- La propriété du composant visuel, en occurrence Text, puis que c'est une zone de texte
- Le nom du DataSet (la source de données)
- Le nom du champ associé (DataTable.Nomcolonne)

```
picturePhoto.DataBindings.Add("image", monDataSet, "ListeEmp.photo", true);
```

Pour assurer une navigation à l'intérieur du DataSet, il faudra faire un BindingContext.

Dans une application Windows, la navigation entre les enregistrements d'une source de données est managée par la couche de liaison de données. **L'objet CurrencyManager** associé à une table ou une vue d'un groupe de données prend en charge une propriété **Position**. La propriété **Position** est utilisée par le mécanisme de liaison de données pour faire en sorte que les différents contrôles lisent et écrivent tous des données dans le même enregistrement.

La propriété Count : Utilisez la propriété Count pour déterminer quand la fin d'une liste a été atteinte. Comme CurrencyManager maintient un tableau d'éléments basé sur 0, la fin de la liste est toujours égale à Count moins 1.

L'objet **BindingContext** parcourt les valeurs des propriétés des différents objets CurrencyManager, comme leur propriété **Position**.

```
this.BindingContext[monDataSet, "ListeEtudiants"].Position += 1;
```

ou comme suit

```
CurrencyManager cm;
cm = (CurrencyManager)BindingContext[monDataSet,"ListeEtudiants"];
cm.Position++;
```

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)

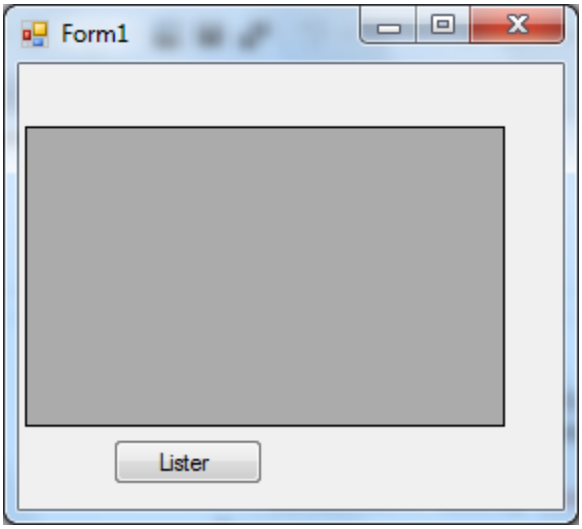
```
BindingSource maSource = new BindingSource(monDataSet, "ResultatEtudiants");
DGVEtudiants.DataSource = maSource;
```

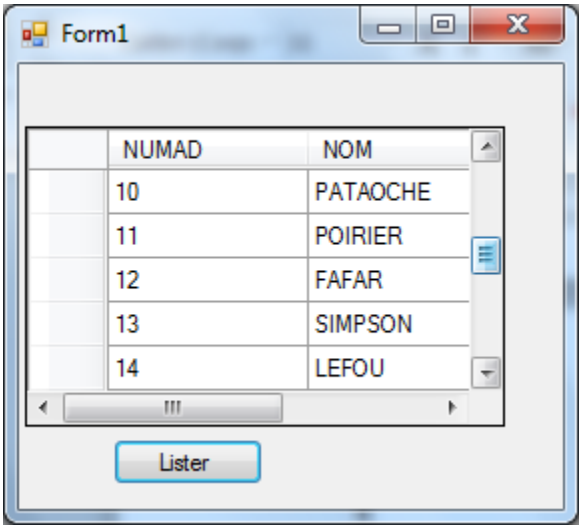
Lorsqu'il s'agit de lier une source de données à une listBox, on utilise la propriété DataSource de celle-ci pour lui affecter les données. La propriété **Displaymember** permet d'afficher le nom de colonne souhaitée

Exemple `string sql = "select nom, prenom from etudiants";`

```
BindingSource uneSource = new BindingSource(monDataSet, "ListeNom");
          listBox1.DataSource = uneSource;
          listBox1.DisplayMember = "nom";
```

11. Exemple utilisant un DataSet





Voici le code qui permet d'afficher dans le DGV :

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

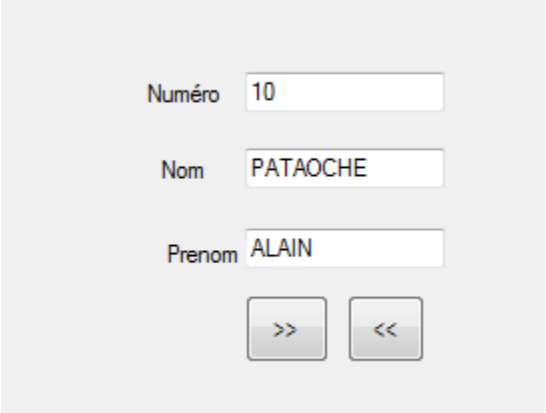
```
namespace WindowsFormDataSet
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

  OracleConnection conn = new OracleConnection();
   private DataSet monDataSet = new DataSet();

  string sql1 = "SELECT NUMAD, NOM, PRENOM FROM ETUDIANTSINFO";
  string sql2 = "select numad, nom, prenom from etudiantsinfo where codep = 420";
```

```
private void afficherDGV()
        {
            try
            {
             OracleDataAdapter Adapter1 = new OracleDataAdapter(sql1, conn);
// On vérifie que le DataSet ne contient pas de Data Table de nom "ListeEtudiants"
                if (monDataSet.Tables.Contains("ListeEtudiants"))
                {
                    monDataSet.Tables["ListeEtudiants"].Clear();
                }
// on rempli le DataSet
            Adapter1.Fill(monDataSet, "ListeEtudiants");
            Adapter1.Dispose();
//on fait une liaison des données entre le DGV et le DataSet pour ListeEtudiants
            BindingSource maSource;
            maSource = new BindingSource(monDataSet, "ListeEtudiants");
            DGVEtudiants.DataSource = maSource;
             }

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

Commentaires :

On utilise un DataSet pour contenir la liste des étudiants.
Le DataSet est rempli par un OracleDataAdapter grâce à la méthode FILL.
La requête SQl est passée à la BD à l'aide de OracleDataAdapter
Le datatable correspondant au résultat s'appelle ResultatEtudiants le résultat est ensuite envoyé dans un DataGridView en utilisant un DataBinding

Utilisation d'un DataBindings sur les zones de Text.



Le code affichant dans les zones de Text est le suivant :

On remplit d'abord leDataSet, puis avec la fonction lier(), on fait les DataBindings.

```
private void afficherTxt()
          {

                try
                 {
              OracleDataAdapter Adapter2 = new OracleDataAdapter(sql2, conn);
               if (monDataSet.Tables.Contains("resEtudiants"))
                     {
                         monDataSet.Tables["resEtudiants"].Clear();
                     }
              Adapter2.Fill(monDataSet, "resEtudiants");
              Adapter2.Dispose();
     // on apelle la fonction lier pour faire la liaison des données
     // du DataSet avec les zones de text.
                lier();
                  }

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

```
private void lier()
      {
   textNumad.DataBindings.Add("text", monDataSet, "resEtudiants.numad");
  textNom.DataBindings.Add("text", monDataSet, "resEtudiants.nom");
  textPrenom.DataBindings.Add("text", monDataSet, "resEtudiants.Prenom");
      }
```

Si vous comptez, utiliser les mêmes contrôles Text pour une autre source de données ou pour la saisie des données, alors il va falloir les dissocier (**ClearBindings**)

```
private void dissocier()
            {
                textNumad.DataBindings.Clear();
                textNom.DataBindings.Clear();
                textPrenom.DataBindings.Clear();
                dateIns.DataBindings.Clear();
                // effacer le contenu
                textNumad.Clear();
                textNom.Clear();
                textPrenom.Clear()
                dateIns.Value = DateTime.Now;
            }
```

Afin de permettre la navigation à l'intérieur de la source de donnée et ce pour tous les contrôle de la forme (ici ce sont tous les TextBox), on utilise un BindingContext

```
// Pour passer à l'enregistrement suivant
private void next_Click(object sender, EventArgs e)
        {
         this.BindingContext[monDataSet, "resEtudiants"].Position += 1;
        }

// Pour passer à l'enregistrement avant
private void previous_Click(object sender, EventArgs e)
        {
         this.BindingContext[monDataSet, "resEtudiants"].Position -= 1;
        }
```

```
// Pour atteindre le premier enregistrement
```

```
private void premier_Click(object sender, EventArgs e)
        {
         this.BindingContext[monDataSet, "resEtudiants"].Position =0;
        }
```

```
// Pour atteindre le dernier enregistrement
```

```
private void dernier_Click(object sender, EventArgs e)
        {
            this.BindingContext[monDataSet, "resEtudiants"].Position =
            this.BindingContext[monDataSet, "resEtudiants"].Count - 1;
        }
```

12. Exemple utilisant un OracleParameter

Soit la forme suivante qui permet d'ajouter un étudiant en utilisant le OracleParameter.

1- La requête utilise une séquence pour insérer des valeurs dans la colonne NUMAD
2- On utilise le contrôle DateTimepicker pour insérer les dates d'inscription.

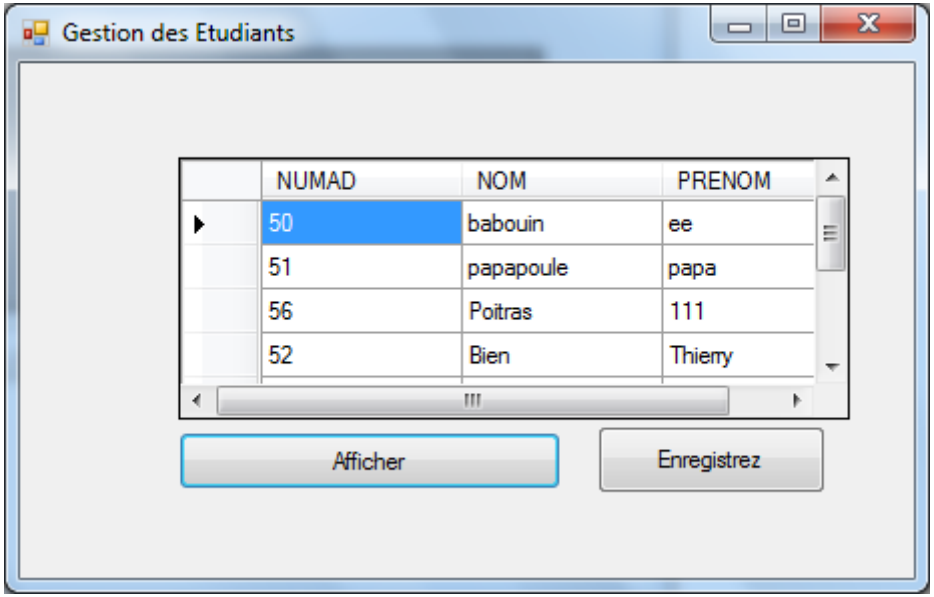
Important :

1- L'objet OracleCommand est utilisé pour passer la requête SQL et les paramètres de la requête.
2- L'ordre dans lequel sont passés les paramètres est important.

```
private void ajouter2_Click(object sender, EventArgs e)
        {
            try
            {
        // la requete SQLajout est paramétrée. Elle a trois paramètres.
        //les paramètres pour Oracle et C # sont précédés de deux points

                string sqlajout = " insert into etudiantsinfo" +
                             " (numad,nom,prenom,dateinscription) values "+
                            "(SeqEtu.nextval,:nom,:prenom,:dateinscription)";

         // On déclare les paramètres pour chaque paramètre de la requête
              // sur une seule ligne.
            OracleParameter oranom = new OracleParameter(":nom",
OracleDbType.Varchar2, 30);

            OracleParameter oraprn = new OracleParameter(":prenom",
OracleDbType.Varchar2, 40);

           OracleParameter oradate = new
OracleParameter(":dateinscription",  OracleDbType.Date);

             // on affecte les valeurs aux paramètres.
                oranom.Value = textNom.Text;
                oraprn.Value = textPrenom.Text;
                oradate.Value = dateIns.Value;

  // on déclare l'objet OracleCommand pour passer la requete
           OracleCommand oraAjout = new OracleCommand(sqlajout, conn);
           oraAjout.CommandType = CommandType.Text;

              // En utilisant la propriété Prameteres de OracleCommand,
               // on spécifie les paramètre de  la requete SQLajout.
                oraAjout.Parameters.Add(oranom);
                oraAjout.Parameters.Add(oraprn);
                oraAjout.Parameters.Add(oradate);
                oraAjout.ExecuteNonQuery();

     // s'il y a un DataBindings sur les zones de Text on appelle la
fonction dissocier pour une autre insertion
 // cette fonction est définie plus haut

                    //dissocier().

            }

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

--suite -

13. Exemple utilisant un DataSet pour les mises à jour :




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

```
namespace ExempleDataSet
{
    public partial class Etudiants : Form
    {
        public Etudiants()
        {
            InitializeComponent();
        }
        private OracleConnection conn = new OracleConnection();
        private DataSet monDataSet = new DataSet();
        private OracleDataAdapter OraAdapter = new OracleDataAdapter();

        private void Etudiants_Load(object sender, EventArgs e)
        {
            try
            {
                string chaineDeconnexion = "Data Source = primogene; User Id = user1; password = user1";
                conn.ConnectionString = chaineDeconnexion;
                conn.Open();
                MessageBox.Show(conn.State.ToString());
            }

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

        }

        private void Quitter_Click(object sender, EventArgs e)
        {
            conn.Close();
            Application.Exit();
        }
```

```
private void lister_Click(object sender, EventArgs e)
      {
          string sql = "select numad,nom, prenom from etudiantsinfo";
          //OracleDataAdapter adapter = new OracleDataAdapter(sql, conn);
          OraAdapter.SelectCommand = new OracleCommand(sql, conn);

          if (monDataSet.Tables.Contains("resEtudiants"))
          {
              monDataSet.Tables["resEtudiants"].Clear();
          }

          // on remplit le DataSet. le nom du data table est resEtudiants
          OraAdapter.Fill(monDataSet, "resEtudiants");
          OraAdapter.Dispose();
          BindingSource masource;
          masource = new BindingSource(monDataSet, "resEtudiants");
          DGVEtudiants.DataSource = masource;

      }



   private void Enregistrer_Click(object sender, EventArgs e)
      {
          string sqlup = "update etudiantsinfo set nom =:nom where numad=:numad";
          string sqlins = "insert into etudiantsinfo (numad,nom,prenom) values(:nuamd,:nom,:prenom)";
          string sqldel = "delete from etudiantsinfo where numad=:numad";


            try
                  {
           //insertion
              OraAdapter.InsertCommand = new OracleCommand(sqlins, conn);
              OraAdapter.InsertCommand.Parameters.Add(":numad", OracleDbType.Int32, 4, "numad");
              OraAdapter.InsertCommand.Parameters.Add(":nom", OracleDbType.Varchar2, 40, "nom");
              OraAdapter.InsertCommand.Parameters.Add(":prenom", OracleDbType.Varchar2, 40, "prenom");
```

```
            // mise à jour du nom
               OraAdapter.UpdateCommand = new OracleCommand(sqlup, conn);
               OraAdapter.UpdateCommand.Parameters.Add(":nom", OracleDbType.Varchar2, 40, "nom");
               OraAdapter.UpdateCommand.Parameters.Add(":numad", OracleDbType.Int32, 4, "numad");

               // suppresion
               OraAdapter.DeleteCommand = new OracleCommand(sqldel, conn);
               OraAdapter.DeleteCommand.Parameters.Add(":numad", OracleDbType.Int32, 4, "numad");

               // execution de la requête
               OraAdapter.Update(monDataSet.Tables["resEtudiants"]);
               monDataSet.AcceptChanges();

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

               finally
                   {
                   OraAdapter.Dispose();
                   }
        }// fin des mies à jours.


    }
}
```

14. ADO.NET et les procédures stockées.

Lorsqu'une procédure ou une fonction est appelée par un programme C# (ADO.NET), nous utilisons les mêmes objets (OracleCommand, OracleDataReader , OracleParameter ..), les mêmes espaces de nom et les même référence.

Il est important d'indiquer au programme, qu'il s'agit d'un appel de procédure stockée (procédure et ou fonction). Vous pouvez indiquer ceci de la manière suivante :

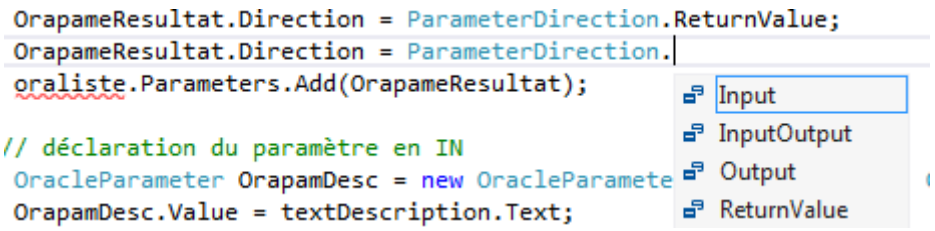
```
OracleCommand oraliste = new OracleCommand("GESTIONPRODUITS", conn);
oraliste.CommandText = "GESTIONPRODUITS.LISTER";
oraliste.CommandType = CommandType.StoredProcedure;
```

GESTIONPRODUITS : indique le nom du Package
GESTIONPRODUITS.LISTER" indique le nom de la procédure stockée à l'interieur du package.
CommandType.StoredProcedure indique que l'on va appeler une procédure stockée.

Il est également important de renseigner le programme de la direction des paramètres de la procédure stockée par la propriété Direction de l'objet OracleParameter. (Voir page 23).

Valeur de la propriété ParameterDirection.

Input	Le paramètre est un paramètre d'entrée. Il s'agit de la valeur par défaut.
Output	Le paramètre est un paramètre de sortie.
ReturnValue	Le paramètre représente une valeur de retour d'une opération telle qu'une procédure stockée (fonction), une fonction intégrée ou une fonction définie par l'utilisateur.
IputOutput	Le paramètre peut être à la fois est un paramètre d'entrée et de sortie.



Le Package GESTIONPRODUITS

```
CREATE OR REPLACE
PACKAGE GESTIONPRODUITS AS
TYPE ENRPRODUIT IS REF CURSOR;

PROCEDURE INSERTION (PNUM IN PRODUITS.NUMP%TYPE,
PDESCRIPTION IN PRODUITS.DESCRIPTION%TYPE,
PPRIX NUMBER);

PROCEDURE CHERCHER(PDESCRIPTION IN PRODUITS.DESCRIPTION%TYPE,
RES OUT ENRPRODUIT);

FUNCTION LISTER(PDESCRIPTION IN VARCHAR2) RETURN ENRPRODUIT;

FUNCTION LISTER2 RETURN ENRPRODUIT;

PROCEDURE MISEAJOUR(PNUM IN NUMBER, PPRIX IN NUMBER);

FUNCTION COMPTER RETURN NUMBER;

END GESTIONPRODUITS;
```

```
CREATE OR REPLACE
PACKAGE BODY GESTIONPRODUITS AS

  PROCEDURE INSERTION (PNUM IN PRODUITS.NUMP%TYPE,
  PDESCRIPTION IN PRODUITS.DESCRIPTION%TYPE,PPRIX NUMBER) AS
  BEGIN
  INSERT INTO produits(NUMP, DESCRIPTION, PRIX) VALUES(PNUM,
PDESCRIPTION,PPRIX);
  COMMIT;
  END INSERTION;

  PROCEDURE CHERCHER(PDESCRIPTION IN PRODUITS.DESCRIPTION%TYPE, RES
OUT ENRPRODUIT) AS
  BEGIN
  OPEN RES FOR SELECT NUMP,DESCRIPTION, PRIX FROM PRODUITS
  WHERE DESCRIPTION LIKE PDESCRIPTION||'%';
  END CHERCHER;

  FUNCTION LISTER(PDESCRIPTION IN VARCHAR2) RETURN ENRPRODUIT AS
  RESULTAT ENRPRODUIT;
  BEGIN
  OPEN RESULTAT FOR SELECT NUMP, DESCRIPTION, PRIX FROM PRODUITS
  WHERE DESCRIPTION LIKE PDESCRIPTION||'%';
  RETURN RESULTAT;
  END LISTER;

  FUNCTION LISTER2 RETURN ENRPRODUIT AS
  RESULTAT ENRPRODUIT;
  BEGIN
  OPEN RESULTAT FOR SELECT DESCRIPTION FROM PRODUITS;
  RETURN RESULTAT;
  END LISTER2;

  PROCEDURE MISEAJOUR(PNUM IN NUMBER, PPRIX IN NUMBER) AS
```

```
BEGIN
UPDATE PRODUITS SET PRIX =PPRIX WHERE NUMP = PNUM;
END MISEAJOUR;

FUNCTION COMPTER RETURN NUMBER AS
TOTAL NUMBER;
BEGIN
SELECT COUNT(*) INTO TOTAL FROM PRODUITS;
RETURN TOTAL;
END COMPTER;
END GESTIONPRODUITS;
```

1- Appel de la procédure INSERTION : tous les paramètres sont en Input.

```
private void Inserer_Click(object sender, EventArgs e)
        {
//déclaration de OracleCommand pour appeler la procédure avec la connexion conn.
//et le nom du Package « GestionProduits». Le type de commande est une procédure
 OracleCommand oraAjout = new OracleCommand("GESTIONPRODUITS", conn);
oraAjout.CommandText = "GESTIONPRODUITS.INSERTION";
oraAjout.CommandType = CommandType.StoredProcedure;

//déclaration des paramètres de la procédure. Les paramètres sont en Input..
OracleParameter orapamNum = new OracleParameter("PNUM",
OracleDbType.Int32);
orapamNum.Direction = ParameterDirection.Input;
orapamNum.Value = Numproduit.Text;
oraAjout.Parameters.Add(orapamNum);

OracleParameter orapamnumDesc = new OracleParameter("PDESCRIPTION",
OracleDbType.Varchar2,50);
orapamnumDesc.Direction = ParameterDirection.Input;
orapamnumDesc.Value = Descriptionproduit.Text;
oraAjout.Parameters.Add(orapamnumDesc);

OracleParameter orapamPrix= new OracleParameter("PPRIX",
OracleDbType.Int32);
orapamPrix.Direction = ParameterDirection.Input;
orapamPrix.Value = PrixProduit.Text;
oraAjout.Parameters.Add(orapamPrix);
//Execution de la requête
oraAjout.ExecuteNonQuery();
        }
```

2- Appel de la fonction `ListerProduits`. Elle retourne un REF CURSOR. Elle a un paramètre en Input. On utilise un DataSet

```
private void ListerProduits_Click(object sender, EventArgs e)
      {
 // la fonction Lister a un paramètre en IN et elle retourne un
REFCURSOR
            try
            {
  //déclaration de OracleCommand pour appeler la fonction avec la
connection conn.
        OracleCommand oraliste = new OracleCommand("GESTIONPRODUITS",
conn);
        oraliste.CommandText = "GESTIONPRODUITS.LISTER";
        oraliste.CommandType = CommandType.StoredProcedure;

     // pour une fonction, le paramètre de retour doit être déclaré en
premier.
        OracleParameter OrapameResultat = new
OracleParameter("RESULTAT", OracleDbType.RefCursor);
         OrapameResultat.Direction = ParameterDirection.ReturnValue;
         oraliste.Parameters.Add(OrapameResultat);

     // déclaration du paramètre en IN
        OracleParameter OrapamDesc = new
OracleParameter("PDESCRIPTION", OracleDbType.Varchar2);
        OrapamDesc.Value = textDescription.Text;
        OrapamDesc.Direction = ParameterDirection.Input;
        oraliste.Parameters.Add(OrapamDesc);

// Pour remplir le DataSet, on déclare un OracleDataAdapter pour lequel
 // on passe notre OracleCommand qui contient TOUS les paramètres.

        OracleDataAdapter orAdater = new OracleDataAdapter(oraliste);

         if (monDataSet.Tables.Contains("ListeProduits"))
                {
                    monDataSet.Tables["ListeProduits"].Clear();
                }

            orAdater.Fill(monDataSet, "ListeProduits");
            oraliste.Dispose();
            BindingSource maSource;
            maSource = new BindingSource(monDataSet, "ListeProduits");
            DGVProduits.DataSource = maSource;


      }
```

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

        }
```

3- Appel de la procédure Afficher : Elle a un paramètre Description en IN et un paramètre RES en OUT de type REF CURSOR. On utilise un DataSet.

```
        private void Afficher_Click(object sender, EventArgs e)
         {

             try{
      OracleCommand Oracmd = new OracleCommand("GESTIONPRODUITS", conn);
      Oracmd.CommandText = "GESTIONPRODUITS.CHERCHER";
      Oracmd.CommandType = CommandType.StoredProcedure;

             OracleParameter OraDesc = new
OracleParameter("PDESCRIPTION", OracleDbType.Varchar2);
             OraDesc.Value = textDescription.Text;
             OraDesc.Direction = ParameterDirection.Input;
             Oracmd.Parameters.Add(OraDesc);

             OracleParameter orapamres = new OracleParameter("RES",
OracleDbType.RefCursor);
             orapamres.Direction = ParameterDirection.Output;
             Oracmd.Parameters.Add(orapamres);

             OracleDataAdapter orAdater = new OracleDataAdapter(Oracmd);

             if (monDataSet.Tables.Contains("Produits"))
             {
                 monDataSet.Tables["Produits"].Clear();
             }

             orAdater.Fill(monDataSet, "Produits");
             Oracmd.Dispose();
             Lister();
             }
             catch (Exception se)
             {
                 MessageBox.Show(se.Message.ToString());
             }
}
```

La fonction Lister affiche dans des zones de Text. En utilisant des boutons suivant et précédent.

```
private void Lister()
        {
    Numproduit.DataBindings.Add("Text", monDataSet, "Produits.nump");
    Descriptionproduit.DataBindings.Add("Text", monDataSet,
"Produits.description");
    PrixProduit.DataBindings.Add("Text", monDataSet, "Produits.Prix");
        }
```

4- Appel d'une fonction qui retourne un REF CURSOR et dont le résultat est envoyé dans un OracleDataReader.

```
        private void listeDescription()
    {

            // la fonction Lister2 retourne un REFCURSOR

  OracleCommand oraliste2 = new OracleCommand("GESTIONPRODUITS", conn);
  oraliste2.CommandText = "GESTIONPRODUITS.LISTER2";
  oraliste2.CommandType = CommandType.StoredProcedure;


   OracleParameter OrapameResultat = new OracleParameter("RESULTAT",
OracleDbType.RefCursor);
    OrapameResultat.Direction = ParameterDirection.ReturnValue;
    oraliste2.Parameters.Add(OrapameResultat);

     OracleDataReader Oraread = oraliste2.ExecuteReader();
                while (Oraread.Read())
                {
                    ListeDescription.Items.Add(Oraread.GetString(0));
                    ListeDescription.SelectedIndex = 0;
                }


    }
```

5- Appel de la fonction Compter. Cette fonction retourne un number.

```
private void TotalProduits()
{
            try {
OracleCommand oraCmdTotal = new OracleCommand("GESTIONPRODUITS", conn);
 oraCmdTotal.CommandText = "GESTIONPRODUITS.compter";
 oraCmdTotal.CommandType = CommandType.StoredProcedure;

OracleParameter OrapamTotal = new OracleParameter("total",
OracleDbType.Int32);
 OrapamTotal.Direction = ParameterDirection.ReturnValue;
oraCmdTotal.Parameters.Add(OrapamTotal);

//comme on retourne un INT, on peut utiliser la méthode ExecuteScalar
 //de l'objet OracleCommande
 oraCmdTotal.ExecuteScalar();
 Total.Text= OrapamTotal.Value.ToString();

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

15. Sources et références

Chapitre 3, Intégration des BD à un Langage de haut niveau (ADO.NET, VB.NET) année 2004 par Vincent Echelard et Denis Brunet.

http://docs.oracle.com/cd/B28359_01/appdev.111/b28844.pdf

http://www.oracle.com/technetwork/articles/dotnet/cook-vs08-088541.html

http://www.oracle.com/technetwork/topics/dotnet/index-085703.html

http://msdn.microsoft.com/fr-fr/library/bb469825.aspx

http://www.oracle.com/technetwork/topics/dotnet/index-085703.html

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

http://msdn.microsoft.com/fr-fr/library/ms752347%28v=vs.110%29.aspx

http://msdn.microsoft.com/fr-ca/library/aa983596%28v=vs.71%29.aspx

http://msdn.microsoft.com/fr-fr/library/yy6y35y8%28v=vs.110%29.aspx