

ADO.Net

- Ensemble de classes offrant des services d'accès aux bases de données(SqlServer, Oracle, MySql...)
- Composants principaux
 - **DataSet**: classe qui gère le stockage des données en mémoire en mode déconnecté. cette classe est totalement indépendante de la nature de la source de données.
 - **Data provider**: classe spécifique au type de source de données. Elle gère la connexion, l'extraction, la modification des données en mémoire et la mise à jour de la source de donnée

ADO.Net

DataSet

- Objets principaux:
 - Collection d'objets **DataTable** représentant les données.
 - Collection d'objets **DataRelation** qui représente les relations inter-tables.
- Un DataSet peut être lié à la plupart des controls Winforms et Webforms (binding).

ADO.Net

DataTable

Collection	Objets de la collection	Description
Columns	DataColumn	Contient les méta-données (column name, data type, possibilité de valeurs Null....)
Rows	DataRow	Représente une ligne de données dans la table
Constraints	Constraint	Contraintes sur une ou plusieurs colonnes par exemple (Unique, foreign Key...)
ChildRelations	DataRelation	Objet de liaison entre une clé primaire et une clé étrangère pour 2 tables

ADO.Net

Data Provider

Chaque provider doit contenir les 4 objets suivant :

- **XxxConnection** : Objet qui encapsule la connexion à une source de donnée spécifique
- **XxxCommand** : Objet qui exécute les commandes dans la source de données (Commandes SQL, Procédures stocké...)
- **DataReader** : Fourni un flux en lecture seule pour la source de donnée. Le flux de données est accessible via la méthode **ExecuteReader** de l'objet XxxCommand.
- **XxxDataAdapter** : Permet la connexion d'un Dataset au XxxCommand. Il génère un Dataset et synchronise les changement avec la source de donnée.

ADO.Net

Connexion

- Pré-requis:
 - Librairie (providers ou drivers) spécifique au type de source de données
 - ConnectionString
- Pour ouvrir la connexion utiliser la méthode `Open()` de l'objet `XxxConnection`

ADO.Net

Exemple

```
using System.Data.SqlClient;

namespace BD
{
    class Program
    {
        static void Main(string[] args)
        {
            //Connexion à un BD SQLServer
            string strCn;
            strCn = "User ID=login; Password=pass; Initial Catalog=cat;";
            strCn += "Data Source=nomBD; Connection Timeout=120";
            SqlConnection sqlCn = new SqlConnection(strCn);
            sqlCn.Open();
        }
    }
}
```

ADO.Net

Fermeture de connexion

- Pour fermer la connexion utiliser la méthode Close() ou Dispose() de l'objet XxxConnection

```
//Fermer la connexion vers la base de donnée  
sqlCn.Close();
```

```
// Détruire l'objet de connexion  
sqlCn.Dispose();
```

ADO.Net

xxxCommand

- Pré-requis:
 - Connexion ouverte vers la source de donnée
 - Commande SQL à exécuter
- Exemple:

```
SqlConnection sqlCn = new SqlConnection(strCn);  
sqlCn.Open();  
  
SqlCommand cmCategories = new SqlCommand("SELECT * FROM Categories", sqlCn);  
  
//Fermer la connexion vers la base de donnée  
sqlCn.Close();
```


ADO.Net

Membres de xxxCommand	Description
CommandText	Texte qui définit la commande
CommandType	Texte SQL ou Procédure Stocké...
Connection	Une connexion ouverte vers la source de donnée
ExecuteNonQuery	Exécuter une commande qui n'a pas de valeur de retour, la méthode retourne uniquement le nombre de lignes affectées (update, insert, create, delete...)
ExecuteReader	Exécute une commande et génère un xxxDataReader
ExecuteScalar	Exécuter une commande qui retourne une seule valeur
ExecuteXmlReader	Exécuter une commande qui retourne un résultat XML (objet XmlReader)
Parameters	Si CommandType est une procédure stocké, Parameters stocke les paramètres de la procédure.

ADO.Net

Construction d'un DataSet

- Instancier l'objet (on peut optionnellement donner un nom):

```
DataSet dsNw= new DataSet("Northwind");
```

- Ajouter une table au dataset:

```
DataTable dtEmployees=dsNw.Tables.Add("Employee");
```

ADO.Net

Construction d'un DataSet

- Ajouter des colonnes à la table:

```
DataSet dsNw = new DataSet("Northwind");
DataTable dtEmployees = dsNw.Tables.Add("Employee");

DataColumn colEmployeeID = dtEmployees.Columns.Add("EmployeeID");
colEmployeeID.DataType = System.Int32;
colEmployeeID.AllowDBNull = false;
colEmployeeID.Unique = true;

DataColumn colLastName = dtEmployees.Columns.Add("LastName");
colLastName.DataType = System.String;
colLastName.Unique = false;

DataColumn colFirstName = dtEmployees.Columns.Add("FirstName");
colFirstName.DataType = System.String;
colFirstName.Unique = false;
```

ADO.Net

Construction d'un DataSet

- Insérer des lignes dans la table:

```
DataRow dr = dtEmployees.NewRow();  
dr["EmployeeID"] = 42;  
dr["LastName"] = "Smith";  
dr["FirstName"] = "Bob";  
dtEmployees.Rows.Add(dr);
```
- On peut substituer les lignes précédentes par :

```
dtEmployees.Rows.Add(new Object() {42,  
"Smith", "Bob"});
```

ADO.Net

RowState

- Chaque objet DataRow possède une propriété RowState qui prend une des valeurs suivantes (défini par l'énumération DataRowState):

ADO.Net

RowState

Valeur	Description
Added	La ligne a été ajoutée et AcceptChanges n'a pas été appelé.
Deleted	La ligne a été supprimée à l'aide de la méthode Delete de DataRow.
Detached	La ligne a été créée, mais n'appartient à aucun DataRowCollection. DataRow est dans cet état immédiatement après sa création et avant son ajout à une collection, ou s'il a été supprimé d'une collection.
Modified	La ligne a été modifiée et AcceptChanges n'a pas été appelé.
Unchanged	La ligne n'a pas été modifiée depuis le dernier appel à AcceptChanges

ADO.Net

DataAdapter: Propriétés

- Le DataAdapter expose quelques propriétés qu'il faut définir pour déterminer son comportement.
- Le texte affecté à ces propriétés peut être une requête SQL ou bien un appel à une procédure stocké
- On n'est pas obligé de définir tous les propriétés, par exemple pour avoir un DataAdapter en lecture seule il suffit de définir SelectCommand

ADO.Net

DataAdapter: Propriétés

Propriété	Description
SelectCommand	La requête SQL utilisé pour générer le DataSet
InsertCommand	La requête SQL utilisé pour insérer une ligne dans la source de donnée
UpdateCommand	La requêteSQL utilisé pour mettre à jour la source de donnée
DeleteCommand	La requête utilisé pour supprimer des lignes de la source de donnée

ADO.Net

DataAdapter: Méthodes

Méthode	Description
Fill	Utiliser SelectCommand pour extraire les données de la source de donnée et générer un DataSet
Update	Utiliser UpdateCommand pour changer les données dans la source de donnée

ADO.Net

DataAdapter: Création

- Ajouter le namespace **System.Data.SqlClient** au code
- Déclarer et instancier un objet DataAdapter.
- Déclarer et instancier un objet Connexion
- Déclarer et instancier un objet Command contenant une requête de sélection
- Affecter l'objet Command à la propriété SelectCommand du adapter.

ADO.Net

DataAdapter: Création

```
SqlConnection cn = new SqlConnection(GetConnectionString());
cn.Open();

SqlDataAdapter adapter = new SqlDataAdapter();

SqlCommand cmd = new SqlCommand("SELECT SupplierID, CompanyName FROM dbo.Suppliers;", cn);

adapter.SelectCommand = cmd;

DataSet ds = new DataSet();

DataTable dt = new DataTable("maTable");

ds.Tables.Add(dt);

adapter.Fill(dt);
```

ADO.Net

```
using System.Data.SqlServerCe;

namespace tpado
{
    class Program
    {
        static void Main(string[] args)
        {
            SqlCeConnection sqlCn = new SqlCeConnection(@"Data Source=BD.sdf");
            SqlCeCommand reqEleve = new SqlCeCommand("SELECT * FROM Eleve", sqlCn);

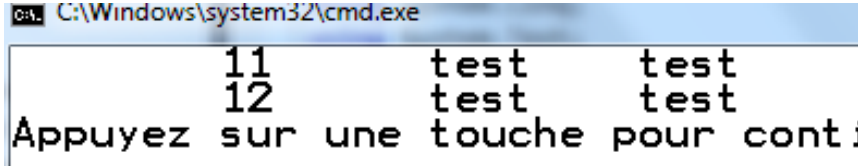
            sqlCn.Open();
            SqlCeDataReader reader = reqEleve.ExecuteReader();

            while (reader.Read())
            {
                Console.WriteLine("\t{0}\t{1}\t{2}", reader[0], reader[1], reader[2]);
            }

            reader.Close();
            sqlCn.Close();
            sqlCn.Dispose();
        }
    }
}
```

SqlServerCe pour Compact Edition (version Express)

reader["id"], reader["Nom"], reader["Prenom"];



```
C:\Windows\system32\cmd.exe
11 test test
12 test test
Appuyez sur une touche pour conti
```

ADO.Net

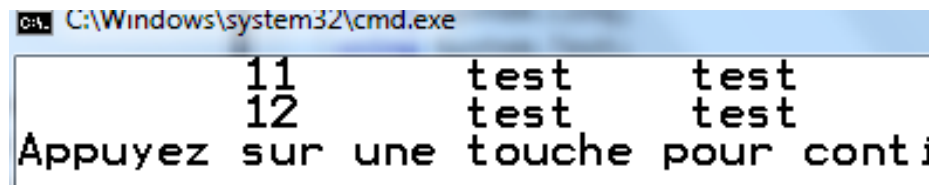
Utiliser using pour se passer de Close et Dispose

```
class Program
{
    static void Main(string[] args)
    {
        using (SqlCeConnection sqlCn = new SqlCeConnection(@"Data Source=BD.sdf"))
        {
            SqlCeCommand reqEleve = new SqlCeCommand("SELECT * FROM Eleve", sqlCn);

            sqlCn.Open();
            SqlCeDataReader reader = reqEleve.ExecuteReader();

            while (reader.Read())
            {
                Console.WriteLine("\t{0}\t{1}\t{2}",
                    reader["id"], reader["Nom"], reader["Prenom"]);
            }

            reader.Close();
        }
    }
}
```



```
C:\Windows\system32\cmd.exe
11 test test
12 test test
Appuyez sur une touche pour conti
```

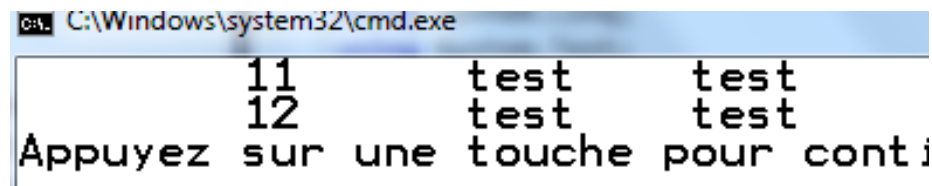
ADO.Net

Encore une fois pour le close du reader

```
class Program
{
    static void Main(string[] args)
    {
        using (SqlCeConnection sqlCn = new SqlCeConnection(@"Data Source=BD.sdf"))
        {
            SqlCeCommand reqEleve = new SqlCeCommand("SELECT * FROM Eleve", sqlCn);

            sqlCn.Open();

            using (SqlCeDataReader reader = reqEleve.ExecuteReader())
            {
                while (reader.Read())
                {
                    Console.WriteLine("\t{0}\t{1}\t{2}",
                        reader["id"], reader["Nom"], reader["Prenom"]);
                }
            }
        }
    }
}
```

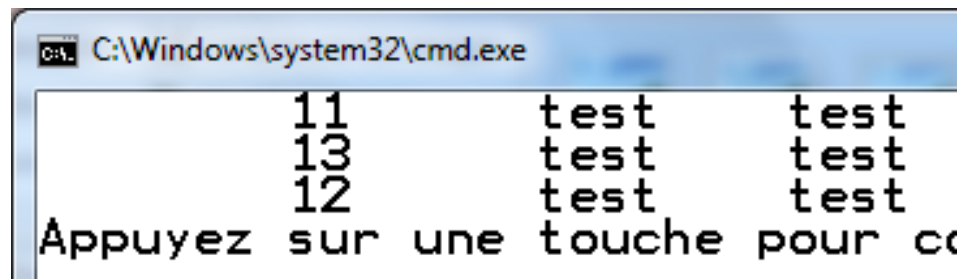


```
C:\Windows\system32\cmd.exe
11      test      test
12      test      test
Appuyez sur une touche pour conti
```

ADO.Net

- Insertion d'une ligne

```
static void Main(string[] args)
{
    using (SqlCeConnection sqlCn = new SqlCeConnection(@"Data Source=BD.sdf"))
    {
        SqlCeCommand reqEleve = new SqlCeCommand("SELECT * FROM Eleve", sqlCn);
        SqlCeCommand insEleve =
            new SqlCeCommand("INSERT INTO Eleve VALUES (13, 'test', 'test')", sqlCn);
        sqlCn.Open();
        insEleve.ExecuteNonQuery();
        using (SqlCeDataReader reader = reqEleve.ExecuteReader())
        {
            while (reader.Read())
            {
                Console.WriteLine("\t{0}\t{1}\t{2}",
                    reader["id"], reader["Nom"], reader["Prenom"]);
            }
        }
    }
}
```



```
C:\Windows\system32\cmd.exe
11      test      test
13      test      test
12      test      test
Appuyez sur une touche pour c
```

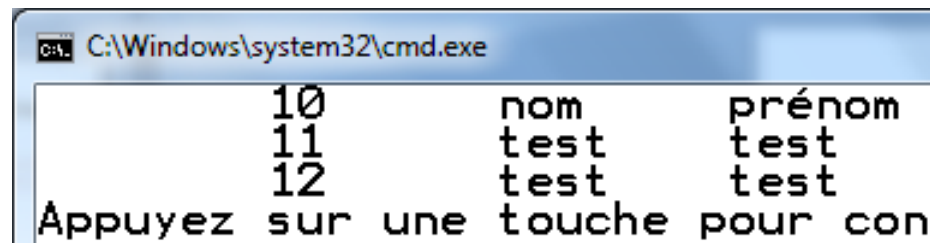
ADO.Net

- Requête paramétré

```
static void Main(string[] args)
{
    using (SqlCeConnection sqlCn = new SqlCeConnection(@"Data Source=BD.sdf"))
    {
        SqlCeCommand reqEleve = new SqlCeCommand("SELECT * FROM Eleve ORDER BY id", sqlCn);
        SqlCeCommand insEleve =
            new SqlCeCommand("INSERT INTO Eleve VALUES (@id, @n, @p)", sqlCn);
        sqlCn.Open();

        insEleve.Parameters.AddWithValue("@id", 10);
        insEleve.Parameters.AddWithValue("@n", "nom");
        insEleve.Parameters.AddWithValue("@p", "prénom");

        insEleve.ExecuteNonQuery();
        using (SqlCeDataReader reader = reqEleve.ExecuteReader())
        {
            while (reader.Read())
            {
                Console.WriteLine("\t{0}\t{1}\t{2}",
                    reader["id"], reader["Nom"], reader["Prenom"]);
            }
        }
    }
}
```



```
C:\Windows\system32\cmd.exe

      10      nom      prénom
      11      test      test
      12      test      test
Appuyez sur une touche pour con
```


ADO.Net

Exercice:

- Écrire un code qui récupère les informations sur un élève et les insère dans la table Eleve
- Ecrire un code qui donne les information sur l'élève dont l'id est récupéré sur console
- Gérer l'exception d'un id inexistant

LINQ

- LINQ (*Language **IN**tegrated **Q**uery*) :
Ensemble d'extensions du langage permettant de faire des requêtes sur des données en faisant abstraction de leur type.
 - permet d'utiliser facilement un jeu d'instructions supplémentaires afin de filtrer des données, faire des sélections, etc.

LINQ

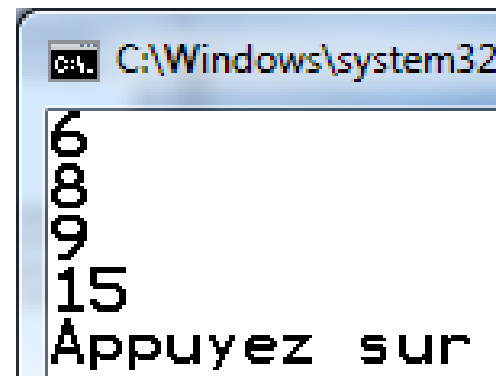
- Il existe plusieurs domaines d'applications pour LINQ :
 - *Linq To Entities* ou *Linq To SQL* qui utilisent ces extensions de langage sur les bases de données.
 - *Linq To XML* qui utilise ces extensions de langage pour travailler avec les fichiers XML.
 - *Linq To Object* qui permet de travailler avec des collections d'objets en mémoire.

LINQ

Exemple : *Linq To Object*

```
class Program
{
    static void Main(string[] args)
    {
        List<int> liste = new List<int> { 4, 6, 1, 9, 5, 15, 8, 3 };
        IEnumerable<int> requete = from i in liste
                                   where i > 5
                                   orderby i
                                   select i;

        foreach (int i in requete)
        {
            Console.WriteLine(i);
        }
    }
}
```



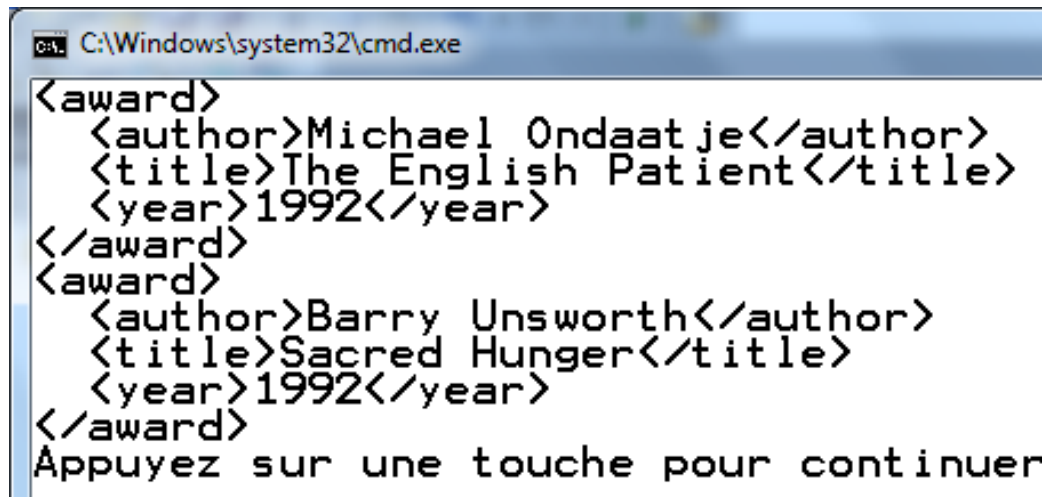
LINQ

Exemple : *Linq To XML*

```
using System.Xml.Linq;

namespace lto
{
    class Program
    {
        static void Main(string[] args)
        {
            XElement livres = XElement.Load(@"c:\booker.xml");
            IEnumerable<XElement> reqLivres = from livre in livres.Descendants("award")
                                              where livre.Element("year").Value.Equals("1992")
                                              select livre;

            foreach (XElement i in reqLivres)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```



```
C:\Windows\system32\cmd.exe
<award>
  <author>Michael Ondaatje</author>
  <title>The English Patient</title>
  <year>1992</year>
</award>
<award>
  <author>Barry Unsworth</author>
  <title>Sacred Hunger</title>
  <year>1992</year>
</award>
Appuyez sur une touche pour continuer
```