C# : ADO.NET (SQL Server)

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

ADO.NET

ADO.NET : **A**ctiveX **D**ata **O**bjects

- Un module (ensemble de classes) dans **.NET Core**.
- Exposant les **services** d'accès et de gestion de données situées dans une base de données relationnelle (**SQL Server**, **MySQL**...) ou **NoSQL**.
- Constitué de deux composants : un premier pour la connexion et un second pour la gestion.

ADO.NET

Deux modes de fonctionnement

- **Mode connecté**
  - La connexion est permanente.
  - Les données à manipuler sont toujours à-jour.
  - Pas de données en mémoire.
- **Mode déconnecté**
  - Le programme se connecte à la base de données, récupère les données et les stocker en mémoire et referme immédiatement la connexion.
  - Les opérations sur les données (affichage...) ne se font qu'une fois la connexion fermée.

ADO.NET

Quelques classes de **ADO.NET** utilisées quel que ce soit le mode

- `SqlConnection` : permet d'assurer la connectivité avec une source de données.
- `SqlCommand` : permet de définir des requêtes **SQL** afin de lire ou écrire de donnée, ou d'exécuter des procédures stockées...
- ...

ADO.NET

Quelques classes de **ADO.NET** utilisées en mode connecté

- `SqlDataReader` : fournit un flux très performant de données en provenance de la source de données.
- `SqlParameter` : associe une valeur à un paramètre de requête.
- ...

ADO.NET

Quelques classes de **ADO.NET** utilisées en mode déconnecté

- `SqlDataSet` : est conçu pour stocker des données venant d'une source de données en mémoire. Il est composé de :
  - `DataTable` : objet correspondant à une table d'une base de données relationnelle. Il contient les objets suivants :
    - `DataColumn` : représente une colonne de la table.
    - `DataRow` : correspond à un tuple de la table.
  - `DataRelation` : objet correspondant à une relation (association) dans une base de données relationnelle.
- `SqlDataAdapter` : utilise la requête définie dans un objet `SqlCommand` afin de remplir le `DataSet` avec des données (intermédiaire entre `DataSet` et la base de données).

ADO.NET

Avant de commencer

- Lancer **Visual Studio**
- Créer un nouveau projet **C#** (Application Console) nommé `CoursAdoDotNet`

ADO.NET

NuGet

- Un gestionnaire de paquets, par défaut, pour **.NET**
- Open-source et gratuit
- Inclus dans *Visual Studio* depuis 2012
- Utilisable aussi en ligne de commande

ADO.NET

NuGet

- Un gestionnaire de paquets, par défaut, pour **.NET**
- Open-source et gratuit
- Inclus dans *Visual Studio* depuis 2012
- Utilisable aussi en ligne de commande

De quel paquets a t-on besoin ?

Microsoft.Data.SqlClient

ADO.NET

Utiliser `NuGet` pour Télécharger les dépendances

- Faire clic droit sur `Dépendances` dans l'`Explorateur de solution`
- Choisir `Gérer les packages NuGet`
- Aller dans l'onglet `Parcourir` et chercher **Microsoft.Data.SqlClient**
- Choisir la dernière version stable et installer
- Accepter, attendre la fin de l'installation

ADO.NET

Utiliser `NuGet` pour Télécharger les dépendances

- Faire clic droit sur `Dépendances` dans l'`Explorateur de solution`
- Choisir `Gérer les packages NuGet`
- Aller dans l'onglet `Parcourir` et chercher **Microsoft.Data.SqlClient**
- Choisir la dernière version stable et installer
- Accepter, attendre la fin de l'installation

On peut aussi installer les packages depuis un terminal

```
dotnet add package Microsoft.Data.SqlClient
```

ADO.NET

Créer une base de données **SQL Server**

- Clic droit sur le nom du projet dans l'`Explorateur de solutions`
- Aller dans `Ajouter > Nouvel élément`
- Dans `Élément Visual C#`, cliquer sur `Données` et choisir `Base de données basée sur les services`
- Saisir `MaBase` (par exemple) dans `Nom :` puis cliquer sur `Ajouter`

ADO.NET

Créer une table

- Dans l'`Explorateur de solutions`, faire double clic sur `MaBase.mdf` (un `Explorateur de serveurs` qui apparaît)
- Dans `Explorateur de serveurs`, faire un clic droit sur l'option `Tables` du menu `MaBase.mdf`
- Choisir `Ajouter une nouvelle table`
- Dans l'onglet `Conception`, remplacer `[dbo].[Table]` par `[dbo].[Personne]`. Ainsi, on a nommé notre table `Personne`
- Par défaut, cette table a une colonne `id`. Ajouter les colonnes `nom`, `prénom` et `age`.

ADO.NET

Créer une table

- Pour que la clé primaire soit **Auto-Increment**, aller dans l'onglet `Propriétés` (le panneau à droite), ouvrir `Spécifications du compteur` ensuite mettre à `True` la valeur de `(Est d'identité)` et vérifier que le compteur et le pas sont à 1.
- Pour exécuter le script, cliquer sur `Mettre à jour`
- Valider en cliquant sur `Mettre à jour la base de données`
- Pour vérifier que la base de données a bien été créée, cliquer à gauche sur `Explorateur de serveurs`, ensuite cliquer sur `Actualiser`, déplier le menu `Tables` où on trouvera la table `Personne`

ADO.NET

Ajouter quelques tuples dans la table `Personne`

- Faire un clic droit sur le nom de la table et choisir `Afficher les données de la table`
- Ajouter quelques tuples sans renseigner la clé primaire (qui est Auto-Increment)
- Aller dans l'`Explorateur de serveurs`, faire un clic droit sur `MaBase` et cliquer sur `Fermer la connexion`

ADO.NET

Les étapes pour faire le CRUD

- Préparer la chaîne de connexion
- Établir la connexion
- Préparer la commande
- Exécuter la commande (et récupérer le résultat)

ADO.NET

Construire la chaîne de connexion

```
SqlConnectionStringBuilder str = new SqlConnectionStringBuilder();
str.AttachDBFilename = "|DataDirectory|\\MaBase.mdf";
str.IntegratedSecurity = true;
str.DataSource = "(LocalDB)\\MSSQLLocalDB";
var connectionString = str.ToString();
```

ADO.NET

Construire la chaîne de connexion

```
SqlConnectionStringBuilder str = new SqlConnectionStringBuilder();
str.AttachDBFilename = "|DataDirectory|\\MaBase.mdf";
str.IntegratedSecurity = true;
str.DataSource = "(LocalDB)\\MSSQLLocalDB";
var connectionString = str.ToString();
```

Ou

```
var connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;
   AttachDbFilename=|DataDirectory|\\MaBase.mdf;Integrated Security=
   True";
```

ADO.NET

Établir la connexion

```
SqlConnection connection = new SqlConnection(connectionString);
connection.Open();
```

ADO.NET

Établir la connexion

```
SqlConnection connection = new SqlConnection(connectionString);
connection.Open();
```

Ou en utilisant `using` pour assurer une fermeture automatique de la connexion

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
        connection.Open();
}
```

ADO.NET

Préparer la commande

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
        connection.Open();
        string queryString = "SELECT * FROM personne";
        SqlCommand command = new SqlCommand(queryString, connection);
}
```

ADO.NET

Préparer la commande

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
        connection.Open();
        string queryString = "SELECT * FROM personne";
        SqlCommand command = new SqlCommand(queryString, connection);
}
```

Exécuter la commande et récupérer le résultat (lecture)

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
        connection.Open();
        string queryString = "SELECT * FROM personne";
        SqlCommand command = new SqlCommand(queryString, connection);
        using (SqlDataReader reader = command.ExecuteReader())
        {
                while (reader.Read())
                {
                        Console.WriteLine($"{reader[0]} : {reader[1]} {reader[2]}");
                }
        }
}
```

ADO.NET

Quelques méthodes de la classe **SqlCommand**

- `ExecuteNonQuery` : permet d'exécuter des requêtes **SQL** ou des procédures stockées qui n'ont pas de valeurs de retour.
- `ExecuteReader` : permet d'exécuter des requêtes **SQL** et de retourner le résultat sous forme d'un tableau de données.
- `ExecuteScalar` : permet d'exécuter des requêtes **SQL** ou des procédures et de retourner une valeur unique.
- `ExecuteXMLReader` : permet de retourner les données sous format **XML**.
- ...

Une deuxième solution (plus verbeuse) avec `SqlDataAdapter` et `DataSet`

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    string queryString = "SELECT * FROM personne";
    SqlCommand command = new SqlCommand(queryString, connection);

    SqlDataAdapter adapter = new SqlDataAdapter(command);
    DataSet myDataSet = new DataSet();
    adapter.Fill(myDataSet, "Personne");
    DataTable dataTable = myDataSet.Tables["Personne"];
    foreach (DataRow row in dataTable.Rows)
    {
        Console.WriteLine(row["id"] + " " + row["nom"] + " " + row["
           prenom"]);
    }
}
```

Une deuxième solution (plus verbeuse) avec `SqlDataAdapter` et `DataSet`

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    string queryString = "SELECT * FROM personne";
    SqlCommand command = new SqlCommand(queryString, connection);

    SqlDataAdapter adapter = new SqlDataAdapter(command);
    DataSet myDataSet = new DataSet();
    adapter.Fill(myDataSet, "Personne");
    DataTable dataTable = myDataSet.Tables["Personne"];
    foreach (DataRow row in dataTable.Rows)
    {
        Console.WriteLine(row["id"] + " " + row["nom"] + " " + row["
           prenom"]);
    }
}
```

L'espace de noms `System.Data` est nécessaire pour cette deuxième solution

```
using System.Data;
```

ADO.NET

DataSet VS DataReader

- DataReader : fonctionne en mode connecté.
- DataSet : fonctionne en mode déconnecté.
- DataReader : plus rapide
- DataSet : explicitement conçu pour un accès aux données indépendamment de toute source de données (XML, Base de données...)

ADO.NET

Insérer une personne dans la base de données

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    string queryString = "INSERT INTO personne (nom, prenom, age)
       VALUES ('wick', 'john', 45)";
    SqlCommand command = new SqlCommand(queryString, connection);
    int i = command.ExecuteNonQuery();
    Console.WriteLine($"number of added persons is {i}");

    queryString = "SELECT * FROM personne";
    command = new SqlCommand(queryString, connection);
    using (SqlDataReader reader = command.ExecuteReader())
    {
        while (reader.Read())
        {
            Console.WriteLine($"{reader[0]} : {reader[1]} {reader[2]}");
        }
    }
}
```

ADO.NET

Attention

- Si on lance le programme plusieurs fois, le même tuple sera inséré une seule fois
- Il y a deux modes
  - **Development** : on travaille donc sur une copie de la base de données, et à chaque lancement, On recharge cette même copie.
  - **Production** : on travaille sur la vraie base de données, si on relance plusieurs fois, le tuple sera ajouté plusieurs fois.

ADO.NET

Comment vérifier ?

- Pour s'assurer, utiliser l'Explorateur de dossiers pour aller dans la racine de votre projet
  - On peut remarquer la présence d'un fichier **MaBase.mdf**. C'est la source utilisée en mode développement.
  - Aller dans `bin/debug`, on peut remarquer aussi la présence d'un deuxième fichier **MaBase.mdf**. C'est le fichier utilisée en mode production.
- Lancer le fichier de votre projet situé dans `bin/debug` plusieurs fois et vérifier que le tuple a aussi été ajouté plusieurs fois

ADO.NET

Une autre problématique

- Si on relance encore une fois le programme à partir de *Visual Studio*, on perd encore les tuples ajoutés.
- Oui, car on a écrasé encore une fois les données.

ADO.NET

Une autre problématique

- Si on relance encore une fois le programme à partir de *Visual Studio*, on perd encore les tuples ajoutés.
- Oui, car on a écrasé encore une fois les données.

Solution

Copier le répertoire `Debug` sur votre bureau (par exemple), et relancé à partir en mode développement ou en mode production, les données seront préservées.

Insérer une personne dans la base de données avec une requête paramétrée

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    string queryString = "INSERT INTO personne (nom, prenom, age)
       VALUES (@nom, @prenom, @age)";

    SqlCommand command = new SqlCommand(queryString, connection);
    command.Parameters.AddWithValue("@nom", "wick");
    command.Parameters.AddWithValue("@prenom", "john");
    command.Parameters.AddWithValue("@age", 45);
    int i = command.ExecuteNonQuery();
    Console.WriteLine($"number of added persons is {i}");

    queryString = "SELECT * FROM personne";
    command = new SqlCommand(queryString, connection);
    using (SqlDataReader reader = command.ExecuteReader())
    {
        while (reader.Read())
        {
            Console.WriteLine($"{reader[0]} : {reader[1]} {reader[2]}");
        }
    }
}
```

Pour récupérer la clé primaire affectée à la dernière personne insérée

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    string queryString = "INSERT INTO personne (nom, prenom, age)
      VALUES (@nom, @prenom, @age); SELECT CAST(scope_identity() AS
      int)";
    SqlCommand command = new SqlCommand(queryString, connection);
    command.Parameters.AddWithValue("@nom", "wick");
    command.Parameters.AddWithValue("@prenom", "john");
    command.Parameters.AddWithValue("@age", 45);
    int i = (Int32)command.ExecuteScalar();
    Console.WriteLine($"Primary key of last inserted person: {i}");

    queryString = "SELECT * FROM personne";
    command = new SqlCommand(queryString, connection);
    using (SqlDataReader reader = command.ExecuteReader())
    {
        while (reader.Read())
        {
            Console.WriteLine($"{reader[0]} : {reader[1]} {reader[2]}");
        }
    }
}
```

ADO.NET

Une deuxième solution avec `SqlDataAdapter` **et** `DataSet`

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    string queryString = "SELECT * FROM personne";
    SqlCommand command = new SqlCommand(queryString, connection);

    SqlDataAdapter adapter = new SqlDataAdapter(command);
    DataSet myDataSet = new DataSet();
    adapter.Fill(myDataSet, "Personne");
    DataTable dataTable = myDataSet.Tables["Personne"];
    var newRow = dataTable.NewRow();
    newRow["age"] = 45;
    newRow["nom"] = "wick";
    newRow["prenom"] = "john";
    dataTable.Rows.Add(newRow);

    using (var builder = new SqlCommandBuilder(adapter))
    {
        adapter.InsertCommand = builder.GetInsertCommand();
        int i = adapter.Update(dataTable);
        Console.WriteLine($"number of added persons is {i}");
    }

    foreach (DataRow row in dataTable.Rows)
    {
        Console.WriteLine(row["nom"] + ",  " + row["prenom"] + ",  " + row["age"]);
    }
}
```

Modifier une personne dans la base de données avec une requête paramétrée

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    string queryString = "UPDATE personne SET nom = @nom, prenom =
       @prenom WHERE id = @id";
    SqlCommand command = new SqlCommand(queryString, connection);
    command.Parameters.AddWithValue("@nom", "denzel");
    command.Parameters.AddWithValue("@prenom", "washington");
    command.Parameters.AddWithValue("@id", 1);
    int i = command.ExecuteNonQuery();
    Console.WriteLine($"number of updated persons is {i}");

    queryString = "SELECT * FROM personne";
    command = new SqlCommand(queryString, connection);
    using (SqlDataReader reader = command.ExecuteReader())
    {
        while (reader.Read())
        {
            Console.WriteLine($"{reader[0]} : {reader[1]} {reader[2]}");
        }
    }
}
```

ADO.NET

Une deuxième solution avec `SqlDataAdapter` **et** `DataSet`

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    string queryString = "SELECT * FROM personne";
    SqlCommand command = new SqlCommand(queryString, connection);

    SqlDataAdapter adapter = new SqlDataAdapter(command);
    DataSet myDataSet = new DataSet();
    adapter.Fill(myDataSet, "Personne");

    DataTable dataTable = myDataSet.Tables["Personne"];
    var selectedRow = dataTable.Rows[0];
    selectedRow["age"] = 65;
    selectedRow["nom"] = "travolta";

    using (var builder = new SqlCommandBuilder(adapter))
    {
        adapter.UpdateCommand = builder.GetUpdateCommand();
        int i = adapter.Update(dataTable);
        Console.WriteLine($"number of updated persons is {i}");
    }

    foreach (DataRow row in dataTable.Rows)
    {
        Console.WriteLine(row["nom"] + ",  " + row["prenom"] + ",  " + row["age"]);
    }
}
```

ADO.NET

Supprimer une personne de la base de données avec une requête paramétrée

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    string queryString = "DELETE FROM personne WHERE id = @id";
    SqlCommand command = new SqlCommand(queryString, connection);

    command.Parameters.AddWithValue("@id", 1);
    int i = command.ExecuteNonQuery();
    Console.WriteLine($"number of deleted persons is {i}");
    queryString = "SELECT * FROM personne";
    command = new SqlCommand(queryString, connection);
    using (SqlDataReader reader = command.ExecuteReader())
    {
        while (reader.Read())
        {
            Console.WriteLine($"{reader[0]} : {reader[1]} {reader[2]}")
                ;
        }
    }
}
```

ADO.NET

Une deuxième solution avec `SqlDataAdapter` **et** `DataSet`

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    string queryString = "SELECT * FROM personne";
    SqlCommand command = new SqlCommand(queryString, connection);

    SqlDataAdapter adapter = new SqlDataAdapter(command);
    DataSet myDataSet = new DataSet();
    adapter.Fill(myDataSet, "Personne");

    DataTable dataTable = myDataSet.Tables["Personne"];
    var selectedRow = dataTable.Rows[0];
    selectedRow.Delete();

    using (var builder = new SqlCommandBuilder(adapter))
    {
        adapter.DeleteCommand = builder.GetDeleteCommand();
        int i = adapter.Update(dataTable);
        Console.WriteLine($"number of deleted persons is {i}");
    }

    foreach (DataRow row in dataTable.Rows)
    {
        Console.WriteLine(row["nom"] + ",  " + row["prenom"] + ",  " + row["age"]);
    }
}
```

ADO.NET

Organisation du code

- Placer toutes les données de connexion (url, nomUtilisateur, motDePasse...) dans une classe de connexion.
- Pour chaque table de la base de données, créer un modèle, une classe **C#**, ayant comme attributs les colonnes de cette table.
- Placer tout le code correspondant à l'accès aux données (de la base de données) dans des nouvelles classes et interfaces qui constitueront la couche **DAO** (Data Access Object).

ADO.NET

La classe `MyConnection` **dans un répertoire** `Configurations`

```
using System;
using System.Data.SqlClient;

namespace CoursAdoDotNet.Configurations
{
    class MyConnection
    {
        private static SqlConnection connection;
        private MyConnection()
        {
            SqlConnectionStringBuilder str = new SqlConnectionStringBuilder();
            str.AttachDBFilename = "|DataDirectory|\\MaBase.mdf";
            str.IntegratedSecurity = true;
            str.DataSource = "(LocalDB)\\MSSQLLocalDB";

            var connectionString = str.ToString();
            connection = new SqlConnection(connectionString);
            try
            {
                connection.Open();
            }
            catch (SqlException e)
            {
                connection = null;
                Console.WriteLine(e.StackTrace);
            }
        }
```

ADO.NET

La classe `MyConnection` (suite)

```
        public static SqlConnection GetConnexion()
        {
            if (connection == null)
            {
                new MyConnection();
            }
            return connection;
        }
        public static void CloseConnection()
        {
            if (connection != null)
            {
                connection.Close();
                connection = null;
            }
        }
    }
}
```

La classe `Personne` dans un répertoire `Models`

```
namespace CoursAdoDotNet.Models
{
    public class Personne
    {
        public int Id { get; set; }
        public String Nom { get; set; }
        public String Prenom { get; set; }
        public int Age { get; set; }

        public Personne()
        {
        }
        public Personne(String nom, String prenom, int age)
        {
            Nom = nom;
            Prenom = prenom;
            Age = age;
        }
        public Personne(int id, String nom, String prenom, int age)
        {
            Id = id;
            Nom = nom;
            Prenom = prenom;
            Age = age;
        }

        public override string ToString()
        {
            return $"Identifiant : { Id }, Nom : { Nom }, Prénom: { Prenom }, Age : { Age }";
        }
    }
}
```

ADO.NET

L'interface `IDao` dans un répertoire `Dao`

```
namespace CoursAdoDotNet.Dao
{
    interface IDao<T>
    {
        T Save(T t);
        T Update(T t);
        void Remove(T t);
        List<T> FindAll();
        T FindById(int id);
    }
}
```

ADO.NET

Créons une classe `PersonneDao` dans le répertoire `Dao`

```
namespace CoursAdoDotNet.Dao
{
    class PersonneDao
    {
    }
}
```

ADO.NET

Faisons hériter la classe `PersonneDao` de l'interface `IDao`

```
namespace CoursAdoDotNet.Dao
{
    class PersonneDao: IDao<Personne>
    {
    }
}
```

Implémentons les méthodes de l'interface `IDao`

```
namespace CoursAdoDotNet.Dao
{
    class PersonneDao : IDao<Personne>
    {
        public List<Personne> FindAll()
        {
            throw new NotImplementedException();
        }

        public Personne FindById(int id)
        {
            throw new NotImplementedException();
        }

        public void Remove(Personne t)
        {
            throw new NotImplementedException();
        }

        public Personne Save(Personne t)
        {
            throw new NotImplementedException();
        }

        public Personne Update(Personne t)
        {
            throw new NotImplementedException();
        }
    }
}
```

ADO.NET

Implémentons la méthode `FindAll()` dans la classe `PersonneDao`

```
namespace CoursAdoDotNet.Dao
{
    public class PersonneDao : IDao<Personne>
    {

        public List<Personne> FindAll()
        {
            SqlDataReader result = null;
            List<Personne> personnes = new List<Personne>();
            SqlConnection connection = MyConnection.GetConnexion();

            if (connection != null)
            {
                string queryString = "SELECT * FROM personne";
                SqlCommand command = new SqlCommand(queryString, connection);
                result = command.ExecuteReader();
                while (result.Read())
                {
                    personnes.Add(new Personne(int.Parse(result[0].ToString()), result[1].
                       ToString(), result[2].ToString(), int.Parse(result[3].ToString())));
                }

            }

            result.Close();
            MyConnection.CloseConnection();
            return personnes;
        }
```

Implémentons la méthode `save()` dans la classe `PersonneDao`

```
        public Personne Save(Personne personne)
        {
            int i = 0;
            SqlConnection connection = MyConnection.GetConnexion();
            if (connection != null)
            {
                string queryString = "INSERT INTO personne (nom, prenom, age) VALUES (@nom,
                   @prenom, @age); SELECT CAST(scope_identity() AS int)";
                SqlCommand command = new SqlCommand(queryString, connection);
                command.Parameters.AddWithValue("@nom", personne.Nom);
                command.Parameters.AddWithValue("@prenom", personne.Prenom);
                command.Parameters.AddWithValue("@age", personne.Age);
                i = (Int32)command.ExecuteScalar();
                personne.Id = i;
                MyConnection.CloseConnection();
            }
            return personne;
        }
        public Personne Update(Personne t)
        {
            throw new NotImplementedException();
        }
        public Personne FindById(int id)
        {
            throw new NotImplementedException();
        }
        public void Remove(Personne t)
        {
            throw new NotImplementedException();
        }
    }
}
```

ADO.NET

Le `Main` pour tester toutes ces classes

```
class Program
{
  static void Main(string[] args)
  {
      PersonneDao personneDao = new PersonneDao();
      Console.WriteLine(personneDao.Save(new Personne("Sy", "
         Omar")));
      List<Personne> personnes = personneDao.FindAll();
      personnes.ForEach(Console.WriteLine);
      Console.ReadKey();
  }
}
```

ADO.NET

Exercice 1

Implémenter les méthodes `Remove`, `Update` et `FindById`

ADO.NET

Exercice 2

Implémenter une méthode `FindAllByPattern(string pattern)` qui retourne toutes les personnes dont le nom ou le prénom contient la chaine `pattern`.