La persistance avec ADO.NET

Définition d'ADO.NET et son rôle dans les applications .NET

Définition d'ADO.NET

- ADO.NET est un modèle de programmation pour les applications de base de données qui fonctionne dans le cadre de .NET Framework
- Il fournit un accès aux données relationnelles à partir de n'importe quelle application .NET
- Il est utilisé pour gérer les connexions, les commandes et les transactions avec une base de données

Rôle d'ADO.NET dans les applications .NET

ADO.NET est un outil clé pour les développeurs .NET pour accéder à une base de données et interagir avec les données

- Il fournit une interface pour les applications pour communiquer avec la base de données
- Il permet de gérer les opérations de base de données telles que la lecture et l'écriture, la mise à jour, la suppression de données, etc.

Les différents types de connexions

ADO.NET fournit plusieurs types de connexions :

- Connexion classique : permet de gérer les connexions de manière traditionnelle à une base de données
- Connexion à pool de connexions : gère un pool de connexions réutilisables
- Connexion distante : permet de gérer les connexions à distance

Connexion à une base de données

Pour se connecter à une base de données à l'aide d'ADO.NET, vous devez utiliser une classe de connexion telle que SqlConnection ou SQLiteConnection

Plusieurs étapes sont nécessaires pour établir une connexion à une base de données :

- Vous devez définir les informations
- Ouverture de la connexion à la base de données
- Fermeture de la connexion à la base de données

Les informations de connexion

Vous devez définir les informations de connexion telles que :

- Le nom de la base de données
- Le nom d'utilisateur
- Le mot de passe (le cas échéant)

Ouverture et fermeture de la connexion

Une fois les informations de connexion définies, vous pouvez ouvrir la connexion à la base de données en utilisant la méthode `Open` de la classe de connexion

Après avoir terminé les opérations de base de données, vous devez fermer la connexion à l'aide de la méthode `Close` de la classe de connexion

Présentation des différents types de commandes :

ADO.NET fournit plusieurs types de commandes pour exécuter des requêtes SQL sur une base de données :

- Commande simple : permet d'exécuter une requête SQL simple sur une base de données
- Commande paramétrée : permet d'exécuter une requête SQL avec des paramètres sur une base de données

Code pour établir une connexion à une base de données SQLite existante :

Dans ADO.NET, la classe SQLiteConnection est utilisée pour établir une connexion à une base de données SQLite existante.

Exemple de code :

```
string connectionString = "Data Source=MonBaseSQLite.db;Version=3;";
SQLiteConnection connection = new SQLiteConnection(connectionString);
connection.Open();
// Code pour exécuter des opérations de base de données
connection.Close();
```

Exécution de requêtes SELECT avec ADO.NET et SQLite :

Création d'une requête SELECT simple :

Une fois la connexion établie, nous pouvons créer une instance de la classe SQLiteCommand et définir la requête SQL que nous souhaitons exécuter à l'aide de la propriété CommandText.

```
// Établissement de la connexion à la base de données
string connectionString = "Data Source=database.db;Version=3;";
SQLiteConnection connection = new SQLiteConnection(connectionString);
connection.Open();

// Création de la commande SQLite
SQLiteCommand command = new SQLiteCommand();
command.Connection = connection;
command.CommandText = "SELECT * FROM customers";
```

Exécution de requêtes SELECT avec des filtres et des tri :

Nous pouvons filtrer les données en utilisant la clause WHERE dans notre requête SQL.

```
command.CommandText = "SELECT * FROM customers WHERE name='John'";
```

Nous pouvons également trier les données en utilisant la clause ORDER BY par exemple :

```
command.CommandText = "SELECT * FROM customers ORDER BY name ASC";
```

Lecture de données à l'aide de la méthode ExecuteReader :

Une fois que nous avons défini notre requête SELECT, nous pouvons lire les données en utilisant la méthode ExecuteReader de la classe SQLiteCommand.

Cette méthode retourne un objet de type SQLiteDataReader qui peut être utilisé pour parcourir les lignes de données retournées par la requête.

Exemple de code pour lire les données à l'aide de la méthode ExecuteReader :

```
SQLiteDataReader reader = command.ExecuteReader();

while (reader.Read())
{
        Console.WriteLine("Name: " + reader["name"] + " | Email: " + reader["email"]);
}

reader.Close();
connection.Close();
```

Manipulation de données avec ADO.NET et SQLite :

CRUD (Create, Read, Update, Delete)

Explication des différentes opérations CRUD :

- Create : Ajout de nouvelles données dans la base de données.
- Read : Lecture des données dans la base de données.
- Update : Mise à jour des données existantes dans la base de données.
- Delete : Suppression des données existantes dans la base de données.

Utilisation de la méthode ExecuteNonQuery pour exécuter des requêtes INSERT, UPDATE et DELETE

Code pour exécuter des requêtes INSERT :

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

        using (SQLiteCommand command = new SQLiteCommand(
                "INSERT INTO table_name (column1, column2, column3)
                 VALUES (@value1, @value2, @value3)",connection))
        {
                command.Parameters.AddWithValue("@value1", "data1");
                command.Parameters.AddWithValue("@value2", "data2");
                command.Parameters.AddWithValue("@value3", "data3");

                command.ExecuteNonQuery();
        }
}
```

Code pour exécuter des requêtes UPDATE :

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

        using (SQLiteCommand command = new SQLiteCommand(
                "UPDATE table_name
                SET column2 = @value2 WHERE column1 = @value1",
                 connection))
        {
                command.Parameters.AddWithValue("@value1", "data1");
                command.Parameters.AddWithValue("@value2", "new_data2");

                command.ExecuteNonQuery();
        }
}
```

Code pour exécuter des requêtes DELETE :

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

        using (SQLiteCommand command = new SQLiteCommand(
                "DELETE FROM table_name WHERE column1 = @value1",
                 connection))
        {
                command.Parameters.AddWithValue("@value1", "data1");

                command.ExecuteNonQuery();
        }
}
```