Objectifs :

- Intégrer les accès aux données dans le client en mode connecté.
- Intégrer les accès aux données dans le client en mode déconnecté.

Pré-requis :

- Connaissances en C#.
- Microsoft Visual Studio.

Plan :

Introduction

L'accès aux données dans le développement d'applications est une étape fondamentale qui influera ensuite sur la rapidité et l'évolutivité de votre application. Dans les versions précédentes de Visual Basic, il existait plusieurs méthodes d'accès aux données en fonction des configurations (Bases de données distantes ou locales, type de fichiers…) :

- DAO (Data Access Object),
- RDO (Remote Data Object),
- ADO (Activex Data Object).
- ADO.Net (Activex Data Object .Net).

L'ancienne version d'ADO pose des problèmes de monté en charge (notamment avec ASP) et de gestion des connexions (connexion ouverte sur la base en permanence).

I) Présentation d'ADO.Net

Avec .NET, Microsoft propose sa dernière technologie d'accès aux données ADO.Net. C'est un ensemble de classes permettant de récupérer et de manipuler des données et qui fait partie intégrante de la nouvelle plate-forme appelée .Net Framework.

1) Les nouveautés d'ADO.Net

Avec ADO.NET Microsoft s'est efforcé de répondre efficacement aux besoins des applications Web en apportant les nouveautés suivantes :

- Gestion du mode déconnecté.
- Introduction des DataSet : Objets "miroir" déconnectés.
- Un meilleur support de XML
- Optimisation des connexions (connexion ouverte uniquement pendant la lecture et l'écriture)
  => Minimiser le temps et les ressources gérés par le serveur.

En mode déconnecté, l'utilisateur travaille sur des données locales qu'il met à jour en bloc lors d'étapes de validation.

L'avantage de ce modèle est qu'il suffit de dériver seulement les classes connectées pour s'interfacer avec une nouvelle base de données. L'ensemble des objets utilisés en mode déconnecté reste le même.

L'accès à la base de données est géré par un ensemble de classes qui constitue le ".NET Data Provider" appelé également fournisseur de données.

2) Fournisseur de données .NET (.NET Data Provider)

ADO.Net supporte 4 fournisseurs par défaut :

- SQL Serveur.Net pour SQL Server v7 et supérieures.
- OleDb.Net (Middleware d'accès à toute une série de bases). Compatibilité VS6.
- ODBC.
- Oracle (version 8.1.7 et supérieures).

D'autres fournisseurs apparaissent sur le net au fur et à mesure du temps (fournisseur MySql, SQLite, PostgreSQL, AS/400, …).

Cette architecture permet alors d'utiliser une syntaxe unique et les mêmes méthodes d'appel quelque soit le fournisseur utilisé :

Seul le type de l'objet créé diffère (et éventuellement aussi les paramètres de la chaîne de connexion).

II) Le modèle d'objet ADO.NET

ADO.Net comprend plusieurs classes permettant l'accès à la base de données.
Deux modes d'accès sont possibles :

<table>
<tr><td rowspan="2">.NET Data Provider :

Classes bas niveaux

Spécifiques à chaque
Type de base

Interfaces
communes</td><td>Accès "classique" par des requêtes SQL ou des procédures stockées qui agissent en direct sur la base
=> Classes Command & DataReader.</td></tr>
<tr><td>Classes associées :
- …Connection : interface IDbConnection
- …Transaction : interface IDbTransaction
- …Command : Interface IDbCommand
- …Parameter : Interface IDbDataParameter
- …DataReader : Interface IDataReader
- …DataAdapter : Interface IDbDataAdapter
Chaque fournisseur implémente ces classes et le "…" est alors remplacé par le préfixe associé au fournisseur.</td></tr>
<tr><td rowspan="2">Accès déconnecté</td><td>Les données en mode déconnecté sont gérées grâce à l'utilisation des DataSet.

La classe DataAdapter permet de faire le lien entre les objets déconnectés et les objets connectés (comme par exemple entre l'objet connection et les objets dataSet/dataTable).</td></tr>
<tr><td>Classes associées :

DataSet : Contient des Tables et des relations. Fonctionne en mode déconnecté.

DataTable : Chaque DataTable contient la description d'une table de la base de données à l'aide des objets DataColumn, DataRow et Constraint.

- DataColumn : Contient la description de chaque colonne. Gère les colonnes calculées.
- DataRow : Contient une ligne de données pour la DataTable associée.
- Constraint : Permet de définir des contraintes locales.
- DataRelation : Décrit les liens entre les tables.</td></tr>
</table>

	- DataView : Permet d'effectuer une vue locale sur un objet DataTable.

Chaque fournisseur de données contient un ensemble de classes dérivées des classes suivantes :

Classe …Connection	Gère la connexion avec la base de données.
Classe …Command	Permet d'exécuter des instructions SQL sur une table de la base. Les commandes exécutées sont séparées en 4 catégories liées à 4 méthodes : ° ExecuteNonQuery pour les requêtes de mises à jour de la base qui ne renvoient rien. ° ExecuteScalar pour les requêtes qui renvoient une seule valeur (renvoie la 1ère colonne de la 1ère ligne). ° ExecuteReader pour les requêtes qui renvoient un groupe de valeurs. Cette méthode renvoie un objet DataReader. ° ExecuteXmlReader : Renvoie un objet XmlReader qui contient des données XML, en lecture seule.
Classe …Parameter	Permet de gérer le passage de paramètres aux requêtes et procédures
Classe …Transaction	Permet de gérer les transactions.
Classe …DataReader	Objet équivalent à un "Recordset" constitué d'un curseur avant en lecture seule. Il est optimisé pour la lecture. Cet objet ne peut être instancié directement et il est forcément créé par la méthode ExecuteReader.

Classe …DataAdapter	Gère les données déconnectées d'un DataSet ou d'un DataTable. Les objets DataAdapter font le lien entre les données déconnectées (DataSet et DataTable) et la base de données à l'aide des 2 méthodes Fill et Update.

Voici quelques exemples des classes dérivées pour les fournisseurs les plus courants :

OleDb.Net	SQL Serveur	ODBC	Oracle
OleDbConnection	sqlConnection	OdbcConnection	OracleConnection
OleDbCommand	sqlCommand	OdbcCommand	OracleCommand
OleDataAdapter	sqlDataAdapter	OdbcDataAdapter	OracleDataAdapter

Une fois la connexion ouverte, il est possible d'agir sur la base soit :

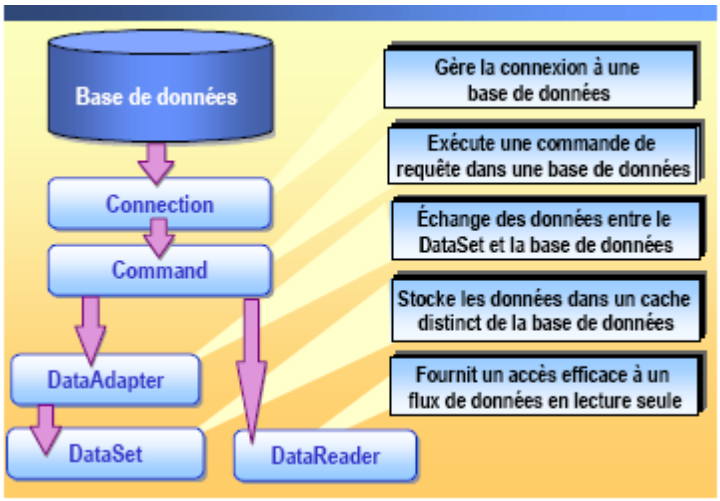
- Avec la classe …Command (ordres SQL directs)
- Par l'association d'un objet de la classe DataAdapter (gestion de l'accès à la base) et d'un DataSet (gestion des données déconnectées).

Pour pouvoir faire appel aux classes proposées par ADO .Net il est nécessaire d'inclure une référence à l'espace de noms correspondant. Vous pouvez soit inclure l'espace System.Data soit inclure des classes de cet espace comme System.Data.OleDb ou System.Data.SqlClient ; tout dépend de la source de données utilisée.

Au final, l'ADO est un ensemble de classe mettant à disposition les objets, méthodes et évènements nécessaire à la connexion avec une base de données. Ces classes sont disponibles dans plusieurs espaces de nom :

Espaces de noms	Description
System.Data	Contient les classes de base de l'architecture ADO.NET permettant de construire des composants capables d'administrer les informations en provenance de sources de données multiples.
System.Data.SqlClient	Contient les classes prenant en charge le fournisseur de données SQL Server de .NET. Ces classes permettent l'accès aux données pour SQL Server 7.0 et supérieur.
System.Data.OleDb	Contient des classes qui prennent en charge le fournisseur de données OLE DB de .NET. Ces classes fournissent un accès managé pour tous les fournisseurs OLE DB pris en charge notamment les SGBD Oracle, Jet, et les versions 6.5 et antérieur de SQL Server

Espaces de nom utilisés pour l'accès aux données



Architecture de ADO.Net

Dans ce qui suit, nous utiliserons le fournisseur de données SqlClient et accèderons à une base SQLServer nommée "*GestionCommandes*" qui contient la table suivante :

Clients *(NumClient, NomClient, Ville, CodePostal)*

II) Manipulation des données avec les objets ADO.NET

1) Mode connecté

On parle de mode connecté lorsque l'application client à un accès direct à la source de données. Dans ce cas, vous devez dans un premier vous connecter à la source avant d'effectuer n'importe quelle tâche. Ce mode est utilisé pour les applications « résidante » au sein d'une entreprise dans laquelle la base de données est toujours accessible.

Dans ce mode, nous allons utiliser les objets suivants Connection, Command et DataReader.

Etapes de manipulation des données en mode connecté :

a) Connexion à la base de données

La connectivité à SQLServer est assurée par l'objet SqlConnection de l'espace de noms **System.Data.SqlClient**.

L'ouverture d'une connexion est réalisée par la méthode **Open** et la fermeture par la méthode **Close**.

L'exemple ci-dessous ouvre une connexion avec un serveur SQL Server :

```
//Exemple de gestion d'une connexion

SqlConnection cn = new SqlConnection();
//Donner une chaîne de connexion pour la propriété ConnectionString
cn.connectionString = "data source=localhost\sqlexpress;initial catalog=
GestionCommandes;integrated security=true;";
//localhost\sqlexpress est le nom du serveur de base de données
//GestioCommandes est le nom de la base de données
cn.open();  //pour ouvrir la connexion

MessageBox.Show(cn.state.toString());//pour afficher l'état de la connexion

cn.close(); //pour fermer la connexion
```

b) Création de la commande

Une fois la connexion établie avec la source de données, vous devez communiquer avec cette dernière pour gérer vos traitements. Trois types de traitements peuvent être effectués :

- Requête de sélection pour extraire des informations
- Requête d'exécution
- Procédures stockées (scripts stockés sur le serveur)

Lors de la création d'un objet commande, vous devez définir le type d'opération qu'il devra réaliser ainsi que la connexion à laquelle il est rattaché.

Propriété	Description
Commandtext	Texte SQL de la requête ou nom de la procédure stockée
CommandType	Type de la commande (requête, table, procédure)
Connection	Connexion liée à la commande
Parameters	Collection de paramètres à envoyer avec la commande

c) Récupération des résultats

De manière générale, il existe deux types de résultat pour un objet command : soit il retourne un seul résultat (c'est le cas lorsque vous utilisez les méthodes ExecuteScalar ou ExecuteNonQuery), soit il retourne un ensemble d'enregistrements (méthode ExecuteReader).

L'objet DataReader permet de lire les enregistrements issus d'une requête renvoyés par l'objet Command.

L'exemple suivant permet de récupérer le résultat d'une requête :

```
SqlCommand myCommand = new SqlCommand();
string req = "select * from Clients";
myCommand.CommandText = req;
myCommand.Connection = cn;
//Ou SqlCommand myCommand = new SqlCommand("select * from Clients", cn);
try {

        SqlDataReader dr = myCommand.ExecuteReader();
        while (dr.Read()) {
          ListBox1.Items.Add(dr[0] + " " + dr[1] + " " + dr[2]);
        }
        dr.Close();
} catch {
        MessageBox.Show("la connexion est fermée");
}
```

L'exemple suivant permet de faire des modifications sur une table :

```
SqlCommand myCommand = new SqlCommand();
myCommand.Connection = cn;
myCommand.CommandText = "Insert Into Clients Values(5,'Adil', 'Fes', 20000)";
myCommand.ExecuteNonQurey();
myCommand.CommandText="Update Clients Set NomClient='Med' where NumClient=5";
myCommand.ExecuteNonQurey();
```

2) Mode déconnecté :

Un environnement déconnecté signifie que les données peuvent être modifiées indépendamment et que les modifications sont écrites dans la base de données ultérieurement.

Avantages :

- Les connexions sont utilisées pour la plus courte durée possible.

Ainsi, un plus petit nombre de connexions peut être utilisé pour un plus grand nombre d'utilisateurs

- Un environnement déconnecté améliore l'évolutivité et les performances des applications

Inconvénients :

- Les données ne sont pas toujours à jour
- Des conflits de modification peuvent survenir et doivent être résolus

Dans ce mode, nous allons utiliser les objets suivants : Connection, Command, DataSet et DataAdapter.

- DataSet

Un DataSet regroupe un ensemble de classe, collections et objets permettant de reproduire une source de données relationnelle avec des objets « Tables » et « Relation ». L'intérêt d'un tel objet est de pouvoir travailler sur des données sans pour autant être connecté à la base.

- DataAdapter

L'objet DataAdapter est un connecteur entre la source de données et l'objet DataSet. L'intérêt d'un tel composant est de pouvoir dissocier la zone de stockage des données de la zone de travail (Le DataSet).

Son utilisation se fait en plusieurs étapes :

- Connexion à la source de données
- Récupération des données de la base et insertion dans les DataTable
- Déconnexion de la source de données
- Modification des données par l'application
- Connexion à la source de données
- Envoi des modifications effectuées vers la source

Etapes de manipulation des données en mode déconnecté :

a) Connexion à la base de données

L'établissement de la connexion se fait de la même manière que le mode connecté :

```
//créer un objet connection

public SqlConnection cn = new SqlConnection("data source =
localhost\\sqlexpress; initial catalog = GestionCommandes; integrated
security = true");
```

b) La lecture de données

La lecture de données se fait en créant un objet DataAdapter.

Lors de la création d'un DataAdapter, il est nécessaire de spécifier la connexion utilisée ainsi que la requête Select. En fonction du Provider utilisé, l'objet DataAdapter sera différent car il doit être en mesure de modifier les données d'une source.

- OleDbDataAdapter
- SqlDataAdapter

Pour créer un objet DataAdapter

- Déclarez l'objet DataAdapter
- Passez une chaîne de requête et un objet **Connection** en tant que paramètres

```
//créer un objet DataAdapter

SqlDataAdapter da = new SqlDataAdapter("select * from Clients", cn);
```

c) La création d'un DataSet

La création d'une instance d'un objet DataSet se fait de la manière suivante :

```
//créer un objet DataSet

DataSet ds = new DataSet();
```

Il est possible de préciser en argument le nom du DataSet, si vous l'omettez c'est le nom "NewDataSet" qui est attribué automatiquement.

d) Charger (et sauver) les données

le remplissage d'un DataSet :
La méthode Fill de l'objet DataAdapter permet d'extraire les données d'une source de données en exécutant la requête SQL spécifiée dans la propriété SelectCommand. Elle prend en paramètre le DataSet et le nom du DataTable à remplir avec les données retournées.

```
da.Fill(ds, "Clients");
```

La mise à jour de données avec un DataAdapter et un DataSet :
La méthode Update de l'objet DataAdapter permet de répercuter sur une source de données les modifications effectuées dans un DataSet. Cette méthode admet un objet DataSet qui contient les données modifiées et un objet DataTable optionnel qui contient les modifications à extraire.

```
da.Update(ds, "Clients");
```

e) Affichage des données

Une fois le DataSet chargé on peut afficher son contenu dans des contrôles comme Les DataGridVeiw, les listeBox ou Combobox

L'affichage se fait en utilisant une boucle sur les lignes du DataSet ou en utilisant la propriété DataSource de ces 3 contrôles

Avec une boucle

```
//Afficher les données sur ListBox
foreach (DataRow ligne in ds.Tables("Clients").Rows) {
      ListBox1.Items.Add(ligne[0] + " " + ligne[1]);
}
//Afficher les données sur ComboBox
foreach (DataRow ligne in ds.Tables("Clients").Rows) {
      ComboBox1.Items.Add(ligne[0]);
}
```

Avec la propriété DataSource

```
//Afficher les données sur ListBox
      ListBox1.DataSource = ds.Tables("Clients");
      ListBox1.DisplayMember = "Nom Colonne";
//Afficher les données sur ComboBox
      Comobox1.DataSource = ds.Tables("Clients");
      Comobox1.DisplayMember = "Nom Colonne";
//Afficher les données sur DataGrid
      DataGridView1.DataSource = ds.Tables("Clients");
```

- Se positionner sur un enregistrement

```
ds.Tables("Clients").Rows[Numero de la ligne][Nom ou indice de la colonne];
```

f) Mise à jour des données

- Ajout

Pour ajouter un enregistrement on crée une ligne (Un objet DataRow) qui a le même schéma que notre DataTable :

```
DataRow ligne = ds.Tables("Clients").NewRow();

//On construit notre ligne
ligne["NumClient"] = 12;
ligne["NomClient"] = "Ali";

//Ou
ligne["NumClient"]  = TextBox1.Text;

//On ajoute la ligne à notre Table (DataTable)

ds.Tables("Clients").Rows.Add(ligne);

//On génère une requete d'insertion sur le DataAdapter

SqlCommandBuilder cmdbuild = new SqlCommandBuilder(da);

//On valide ensuite l'ajout par un Update sur le DataAdapter

da.update(ds, "Clients");
```

- Modification

Pour modifier une ligne dans le DataSet on doit connaître son indice (numéro).

Pour récrupérer l'indice de la ligne on peut parcourir les différentes lignes à la recherche de la ligne modifiée par l'utilisateur ou bien on crée une variable globale (`RowNumber`) qui pointe sur l'enregistrement courant et qu'il faut changer à chaque ajout ou suppression.

Il existe une méthode plus simple dans le cas où on a un combobox associé à une colonne (NumClient par exemple). Donc il suffit d'utiliser la propriété 'SelectedIndex' du Combobox

Exemple :

```
int i=0;

i = Combobox1.SelectedIndex;

ds.Tables("Clients").Rows[i][0] = TextBox1.Text;

ds.Tables("Clients").Rows[i][1] = TextBox2.Text;

//creation objet builder pour generer les cdts de mis a jour efectuer durant
les traitements ds dataset portees par dataadapter

SqlCommandBuilder cmd = new SqlCommandBuilder(da);

//le renvois des changements a la base donnees

da.Update(ds, " Clients");
```

- Suppression

Pour supprimer une ligne on doit aussi connaître son indice.
Une fois l'indice de la ligne est récupéré (de la même manière que la modification) on appelle la méthode Delete

```
ds.Tables("Clients").Rows[IndiceLigneASupprimer].Delete();
```

g) Recherche

La recherche d'enregistrement se fait de plusieurs manières, cela dépend du nombre et du type d'enregistrements attendus de la requête demandée :

Cas 1 : La requête retourne plusieurs résultats qui seront affichés dans des TextBox ou ComboBox

On utilise la méthode Select de l'objet DataTable qui retourne un tableau de lignes :
Dim lignes as DataRow() = ds.Tables(«Table»).Select(«Expression », « Ordre »)
Le paramètre 'Ordre' est optionnel.

Exemple : Afficher les Clients dont le nom commence par 'A' et le code supérieur à 3

```
DataRow[] lignes = ds.Tables("Clients").Select("NomClient LIKE 'A%' AND numClient>3", "numClient ASC");
//Afficher les données sur ComboBox
foreach (DataRow ligne in lignes) {
      ComboBox1.Items.Add(ligne["NumClient"]);
}
```

Cas 2 : La requête retourne plusieurs résultats qui seront affichés dans un DataGridView :
Dans ce cas, on utilise un objet DataView

La classe DataView permet de créer de véritables vues à partir des données d'un DataSet. Ces vues sont effectuée sur une table donnée :

Exemple

```
DataView dv = new DataView();
dv.Table = ds.Tables("Clients");
dv.RowFilter= "NomClient like 'A%' AND numClient > 3"; //Pour le filtre
dv.Sort = "NumClient Asc";  //Pour le tri
DataGridView1.DataSource = dv;
```

On peut récupérer les lignes de la vue et les afficher dans des TextBox

Exemple :

```
if ((dv.Count > 0)) {
      //recuperation des enregistrements recherchés
      foreach (DataRowView drv in dv) {
            //Affichage dans des TextBox par exemlpe
            TextBox1.Text = drv["numClient"];
            TextBox2.Text = drv["nomClient"];
      }
}
```