100% COURS
MCours.com
Cours et mémoires gratuits
mcourscom@gmail.com

Chapitre 1 : INTRODUCTION

Mode connecté et mode déconnecté

Le traitement des données repose traditionnellement sur un modèle à deux couches utilisant une connexion. Le traitement des données utilisant de plus en plus des architectures multicouches, les programmeurs s'orientent vers une approche déconnectée de façon à proposer une meilleure évolutivité pour leurs applications.

XML et ADO.NET

ADO.NET tire parti de la puissance de XML pour fournir un accès déconnecté aux données. ADO.NET a été conçu avec les classes XML du .NET Framework ; les deux composants appartiennent à une même architecture.
ADO.NET et les classes XML du .NET Framework convergent dans l'objet **DataSet**. Le **DataSet** peut être rempli de données provenant d'une source XML, qu'il s'agisse d'un fichier ou d'un flux XML. Le **DataSet** peut être écrit en XML conforme au W3C (World Wide Web Consortium), y compris son schéma, en tant que schéma en langage XSD (XML Schema Definition), quelle que soit la source des données contenues dans le **DataSet**. Le format de sérialisation natif du **DataSet** étant XML, il constitue un excellent support pour le déplacement de données entre couches, faisant ainsi du **DataSet** le meilleur choix pour proposer un accès distant aux données et au contexte du schéma vers et à partir d'un service Web XML.

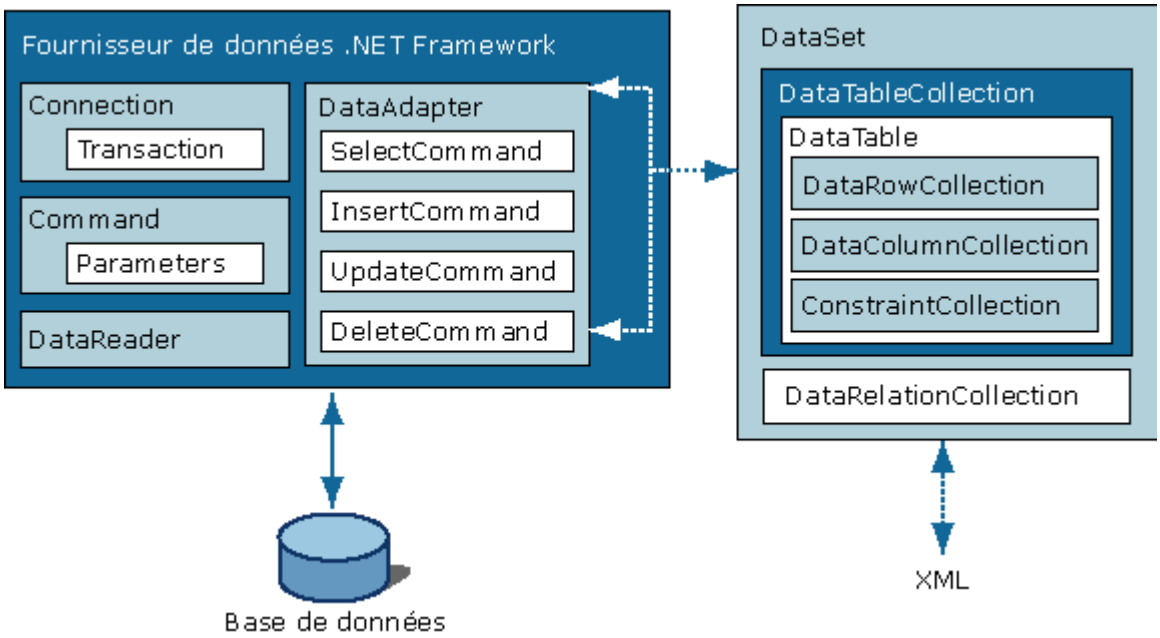
Composants de ADO.NET

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 .NET Framework, qui est un ensemble de composants comprenant les objets **Connection**, **Command**, **DataReader** et **DataAdapter**.
Le DataSet ADO.NET est le composant principal de l'architecture déconnectée de ADO.NET. Le **DataSet** est explicitement conçu pour permettre un accès aux données indépendant de toute source de données. Il peut donc être utilisé avec plusieurs sources de données différentes, utilisé avec des données XML ou utilisé pour gérer des données locales de l'application. Le **DataSet** contient une collection d'un ou de plusieurs

objets **DataTable** constitués de lignes et de colonnes de données, ainsi que des informations concernant les contraintes de clé primaire, de clé étrangère et des informations relationnelles sur les données contenues dans les objets **DataTable**.

L'autre élément principal de l'architecture ADO.NET est **le fournisseur de données .NET Framework** dont les composants sont explicitement conçus pour la manipulation des données et un accès aux données rapide, avant uniquement (forward only) et en lecture seule. 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**.
Vous pouvez écrire des fournisseurs de données .NET Framework pour n'importe quelle source de données. Le .NET Framework est livré avec deux fournisseurs de données .NET Framework : le fournisseur de données .NET Framework pour SQL Server et le fournisseur de données .NET Framework pour OLE DB.
Le schéma suivant représente les composants de l'architecture ADO.NET.

Architecture ADO.NET



Le code suivant montre comment inclure l'espace de noms **System.Data** dans vos applications pour que vous puissiez utiliser ADO.NET.

```
Imports System.Data
```

Les classes ADO.NET se trouvent dans System.Data.dll et sont intégrées aux classes XML de System.Xml.dll. Lors de la compilation du code qui utilise l'espace de noms **System.Data**, il convient de référencer System.Data.dll ainsi que System.Xml.dll.

Connexion à SQL Server à l'aide de ADO.NET

L'exemple de code suivant illustre la création et l'ouverture d'une connexion à une base de données SQL Server (version 7.0 ou ultérieure).

```
Dim nwindConn As SqlConnection = New SqlConnection("Data
Source=localhost;Integrated Security=SSPI;Initial
Catalog=northwind")
nwindConn.Open()
```

Fermeture de la connexion

Il est recommandé de toujours fermer l'objet **Connection** lorsque vous avez fini de l'utiliser. Pour cela, utilisez les méthodes **Close** ou **Dispose** de l'objet **Connection**.

Connexion à une source de données OLE DB à l'aide de ADO.NET

L'exemple de code suivant illustre la création et l'ouverture d'une connexion à une source de données OLE DB.

```
Dim nwindConn As OleDbConnection = New
OleDbConnection("Provider=SQLOLEDB;Data
Source=localhost;Integrated Security=SSPI;Initial
Catalog=northwind")
nwindConn.Open()
```

Connexion à une source de données ODBC à l'aide de ADO.NET

L'exemple de code suivant illustre la création et l'ouverture d'une connexion à une source de données ODBC.

```
Dim nwindConn As OdbcConnection = New
OdbcConnection("Driver={SQL Server};Server=localhost;
Trusted_Connection=yes;Database=northwind")
nwindConn.Open()
```

Connexion à une source de données Oracle à l'aide de ADO.NET

L'exemple de code suivant illustre la création et l'ouverture d'une connexion à une source de données Oracle.

```
Dim nwindConn As OracleConnection = New OracleConnection("Data
Source=MyOracleServer;Integrated Security=yes;")
nwindConn.Open()
```

Exécution d'une commande

Après avoir établi une connexion à une source de données, vous pouvez exécuter des commandes et retourner les résultats de la source à l'aide d'un objet **Command**.

SqlClient

```
Dim catCMD As SqlCommand = New SqlCommand("SELECT CategoryID,
CategoryName FROM Categories", nwindConn)
```

OleDb

```
Dim catCMD As OleDbCommand = New OleDbCommand("SELECT
CategoryID, CategoryName FROM Categories", nwindConn)
```

Obtention d'une valeur unique à partir d'une base de données

Vous aurez peut-être besoin de retourner des informations de base de données qui sont simplement une valeur unique plutôt qu'une table ou un flux de données. Par exemple, vous souhaitez éventuellement retourner le résultat d'une fonction d'agrégation telle que Count(*), Sum(Price) ou Avg(Quantity). L'objet **Command** fournit la fonctionnalité permettant de retourner des valeurs uniques à l'aide de la méthode **ExecuteScalar**. La méthode **ExecuteScalar** retourne comme valeur scalaire la valeur de la première colonne de la première ligne du jeu de résultats.
L'exemple de code suivant retourne le nombre d'enregistrements dans une table utilisant la fonction d'agrégation **Count**.

```
Dim ordersCMD As SqlCommand = New SqlCommand("SELECT Count(*)
FROM Orders", nwindConn)
Dim count As Int32 = CInt(ordersCMD.ExecuteScalar())
```

Exécution d'opérations de catalogue

Pour exécuter une commande permettant de modifier une base de données ou un catalogue, comme l'instruction CREATE TABLE ou CREATE PROCEDURE, créez **Command** à l'aide de la ou des instructions Transact-

SQL appropriées et de **Connection**. Exécutez la commande avec la méthode **ExecuteNonQuery** de l'objet **Command**.
L'exemple de code suivant crée une procédure stockée dans une base de données Microsoft SQL Server.

```
Dim createStr As String = "CREATE PROCEDURE InsertCategory " & _
                          " @CategoryName nchar(15), " & _
                          " @Identity int OUT " & _
                          "AS " & _
                          "INSERT INTO Categories (CategoryName) " & _
                          "VALUES (@CategoryName) " & _
                          "SET @Identity = @@Identity " & _
                          "RETURN @@ROWCOUNT"

Dim createCMD As SqlCommand = New SqlCommand(createStr, nwindConn)
createCMD.ExecuteNonQuery()
```

Modification de données dans une base de données

Les instructions SQL qui modifient les données (comme INSERT, UPDATE ou DELETE) ne retournent pas de lignes. De même, de nombreuses procédures stockées effectuent une action mais ne retournent pas de lignes. Pour exécuter des commandes qui ne retournent pas de lignes, créez un objet **Command** avec la commande SQL appropriée et **Connection** (et les **Parameters** requis) puis utilisez la méthode **ExecuteNonQuery** de l'objet **Command**.

La méthode **ExecuteNonQuery** retourne un entier qui représente le nombre de lignes affectées par l'instruction ou la procédure stockée exécutée. Si plusieurs instructions sont exécutées, la valeur retournée est la somme des enregistrements affectés par toutes ces instructions.
L'exemple de code suivant exécute une instruction INSERT pour insérer un enregistrement dans une base de données à l'aide de **ExecuteNonQuery**.

```
Dim nwindConn As SqlConnection = New SqlConnection("Data
Source=localhost;Integrated Security=SSPI;Initial Catalog=northwind")
nwindConn.Open()

Dim insertStr As String = "INSERT INTO Customers (CustomerID, CompanyName)
Values('NWIND', 'Northwind Traders')"

Dim insertCMD As SqlCommand = New SqlCommand(insertStr, nwindConn)
Dim recordsAffected As Int32 = insertCMD.ExecuteNonQuery()
```

L'exemple de code suivant exécute la procédure stockée créée par l'exemple de code dans Exécution d'opérations de catalogue. Aucune ligne n'est retournée par la procédure stockée, la méthode **ExecuteNonQuery** est donc utilisée mais la procédure stockée reçoit un paramètre d'entrée et retourne un paramètre de sortie et une valeur de retour.
Pour l'objet **OleDbCommand**, le paramètre **ReturnValue** doit être d'abord ajouté à la collection **Parameters**.

```
Dim insertCatCMD As SqlCommand = New SqlCommand("InsertCategory" , nwindConn)
```

```
insertCatCMD.CommandType = CommandType.StoredProcedure

Dim workParm As SqlParameter

workParm = insertCatCMD.Parameters.Add("@RowCount", SqlDbType.Int)
workParm.Direction = ParameterDirection.ReturnValue

workParm = insertCatCMD.Parameters.Add("@CategoryName", SqlDbType.NChar, 15)

workParm = insertCatCMD.Parameters.Add("@Identity", SqlDbType.Int)
workParm.Direction = ParameterDirection.Output

insertCatCMD.Parameters("@CategoryName").Value = "New Category"
insertCatCMD.ExecuteNonQuery()

Dim catID As Int32 = CInt(insertCatCMD.Parameters("@Identity").Value)
Dim rowCount As Int32 = CInt(insertCatCMD.Parameters("@RowCount").Value)
```

Chapitre 2 : UTILISATION DE DATAREADER

Vous pouvez utiliser le **DataReader** ADO.NET pour extraire d'une base de données un flux de données en lecture seule et avant uniquement. 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'utilisation de **DataReader** peut augmenter les performances de l'application d'abord en extrayant les données dès qu'elles sont disponibles, plutôt que d'attendre que tous les résultats de la requête aient été retournés, et ensuite en ne stockant (par défaut) qu'une seule ligne à la fois dans la mémoire, ce qui réduit la charge du système.
Après avoir créé une instance de l'objet **Command**, vous créez un **DataReader** en appelant **Command.ExecuteReader** pour extraire les lignes d'une source de données, comme le montre l'exemple suivant.

```
Dim myReader As SqlDataReader = myCommand.ExecuteReader()
```

Vous utilisez la méthode **Read** de l'objet **DataReader** pour obtenir une ligne des résultats de la requête. Vous pouvez accéder à chaque colonne de la ligne retournée en passant le nom ou la référence ordinale de la colonne au **DataReader**. Cependant, pour une meilleure performance, le **DataReader** fournit une série de méthodes qui vous permettent d'accéder aux valeurs de colonnes dans leurs types de données natifs (**GetDateTime**, **GetDouble**, **GetGuid**, **GetInt32**, etc.).
Remarque La version 1.1 du .NET Framework inclut une propriété supplémentaire pour le **DataReader**, **HasRows**, laquelle vous permet de déterminer si le **DataReader** a retourné des résultats avant de le lire.
L'exemple de code suivant itère dans un objet **DataReader** et retourne une colonne à partir de chaque ligne.

```
Module Module1

    Sub Main()
        Dim cn As New OleDb.OleDbConnection()
       cn.ConnectionString ="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\bd.mdb;"
        cn.Open()

        Dim CMD As New OleDb.OleDbCommand("select * from Article", cn)
        Dim RDR As OleDb.OleDbDataReader = CMD.ExecuteReader()
        While (RDR.Read())
            Console.WriteLine(RDR.GetString(1))
        End While
        Console.ReadLine()
        RDR.Close()
        cn.Close()
    End Sub

End Module
```

Le **DataReader** fournit un flux de données non mis en tampon qui permet à la logique procédurale de traiter efficacement les résultats provenant d'une source de données de façon séquentielle. Le **DataReader** se révèle être un bon choix lors de l'extraction de grandes quantités de données car celles-ci ne sont pas mises en cache dans la mémoire.

Fermeture du DataReader
Vous devez toujours appeler la méthode **Close** lorsque vous avez fini d'utiliser l'objet **DataReader**.
Si **Command** contient des paramètres de sortie ou des valeurs de retour, ils ne seront pas disponibles avant la fermeture du **DataReader**.
Notez que pendant l'ouverture d'un **DataReader**, **Connection** est utilisé en mode exclusif par ce **DataReader**. Vous ne pourrez pas exécuter les commandes pour **Connection**, y compris la création d'un autre **DataReader**, jusqu'à la fermeture du **DataReader** d'origine.

Chapitre 3 : UTILISATION DES PROCEDURES STOCKEES AVEC UNE COMMANDE

Exécution d'une procédure stockée

Les procédures stockées offrent de nombreux avantages dans les applications pilotées par des données. En utilisant les procédures stockées, les opérations de base de données peuvent être encapsulées dans une commande unique, optimisées pour une meilleure performance et améliorées grâce à une sécurité supplémentaire. Tandis qu'une procédure stockée peut être appelée en passant simplement son nom suivi des arguments de paramètre comme instruction SQL, l'utilisation de la collection **Parameters** de l'objet **Command** ADO.NET vous permet de définir plus explicitement les paramètres de procédure stockée et d'accéder aux paramètres de sortie et aux valeurs de retour.
Pour appeler une procédure stockée, affectez **StoredProcedure** au **CommandType** de l'objet **Command**. Une fois **StoredProcedure** affecté à **CommandType**, vous pouvez utiliser la collection **Parameters** pour définir les paramètres, comme dans l'exemple suivant.

Ce code appelle la procédure stockée « VentesParCatégorie » qui accepte un paramètre en entrée.

```
Dim nwindConn As SqlConnection = New SqlConnection("Data
Source=localhost;Integrated Security=SSPI;Initial Catalog=northwind")

Dim salesCMD As SqlCommand = New SqlCommand("VentesParCatégorie",
nwindConn)
salesCMD.CommandType = CommandType.StoredProcedure

Dim myParm As SqlParameter = salesCMD.Parameters.Add("@Categorie",
SqlDbType.NVarChar, 15)
myParm.Value = "Condiments"

nwindConn.Open()

Dim myReader As SqlDataReader = salesCMD.ExecuteReader()

Console.WriteLine(myReader.GetName(0), myReader.GetName(1))

Do While myReader.Read()
  Console.WriteLine(myReader.GetString(0), myReader.GetDecimal(1))
Loop

myReader.Close()
nwindConn.Close()
```

Utilisation des paramètres

Un objet **Parameter** peut être créé à l'aide du constructeur **Parameter** ou en appelant la méthode **Add** de la collection **Parameters** de **Command**. **Parameters.Add** prendra comme entrée les arguments de constructeur ou un objet **Parameter** existant. Lorsque vous affectez une référence null au **Value** d'un **Parameter**, utilisez **DBNull.Value**.
Pour les paramètres autres que **Input**, vous devez définir la propriété **ParameterDirection** pour spécifier si le type de paramètre est **InputOutput**, **Output ou ReturnValue**. L'exemple suivant illustre la différence entre la création des paramètres Input, Output et ReturnValue.

```
Dim sampleCMD As SqlCommand = New SqlCommand("SampleProc", nwindConn)
sampleCMD.CommandType = CommandType.StoredProcedure

Dim sampParm As SqlParameter = sampleCMD.Parameters.Add("RETURN_VALUE",
SqlDbType.Int)
sampParm.Direction = ParameterDirection.ReturnValue

sampParm = sampleCMD.Parameters.Add("@InputParm", SqlDbType.NVarChar, 12)
sampParm.Value = "Sample Value"

sampParm = sampleCMD.Parameters.Add("@OutputParm", SqlDbType.NVarChar, 28)
sampParm.Direction = ParameterDirection.Output

nwindConn.Open()

Dim sampReader As SqlDataReader = sampleCMD.ExecuteReader()

Console.WriteLine(sampReader.GetName(0), sampReader.GetName(1))

Do While sampReader.Read()
  Console.WriteLine(sampReader.GetInt32(0), sampReader.GetString(1))
Loop

sampReader.Close()
nwindConn.Close()

Console.WriteLine(" @OutputParm: ", sampleCMD.Parameters("@OutputParm").Value)
Console.WriteLine("RETURN_VALUE: ", sampleCMD.Parameters("RETURN_VALUE").Value)
```

Utilisation des paramètres avec SqlCommand

Lorsque des paramètres sont utilisés avec **SqlCommand**, les noms des paramètres ajoutés à **SqlParameterCollection** doivent correspondre à ceux des marqueurs de paramètre de votre procédure stockée. Le fournisseur de données .NET Framework pour SQL Server traite les paramètres de la procédure stockée comme des paramètres nommés et recherche les marqueurs de paramètre correspondants.
Le fournisseur de données .NET Framework pour SQL Server ne prend pas en charge l'espace réservé de point d'interrogation (?) pour le passage des paramètres à une instruction SQL ou une procédure stockée. Dans ce cas, vous devez utiliser des paramètres nommés, comme dans l'exemple suivant.

```
SELECT * FROM Customers WHERE CustomerID = @CustomerID
```

Utilisation des paramètres avec OleDbCommand ou OdbcCommand

Lorsque des paramètres sont utilisés avec **OleDbCommand** ou **OdbcCommand**, l'ordre des paramètres ajoutés à **Parameters** doit correspondre à celui des paramètres définis dans votre procédure stockée. Les fournisseurs de données .NET Framework pour OLE DB et ODBC traitent les paramètres d'une procédure stockée comme des espaces réservés et applique des valeurs de paramètre par ordre. En outre, les paramètres des valeurs de retour doivent être les premiers ajoutés à la collection **Parameters**.
Les fournisseurs de données .NET Framework pour OLE DB et ODBC ne prennent pas en charge les paramètres nommés pour le passage des paramètres à une instruction SQL ou une procédure stockée. Dans ce cas, vous devez utiliser l'espace réservé de point d'interrogation (?), comme dans l'exemple suivant.

```
SELECT * FROM Customers WHERE CustomerID = ?
```

En conséquence, l'ordre dans lequel les objets **Parameter** sont ajoutés à la collection **Parameters** doit directement correspondre à la position de l'espace réservé de point d'interrogation pour le paramètre.

Chapitre 4 : REMPLISSAGE D'UN DATASET A PARTIR D'UN DATAADAPTER

Le DataSet

Le **DataSet** ADO.NET est une représentation de données résident en mémoire qui propose un modèle de programmation relationnel cohérent indépendant de la source de données. Le **DataSet** représente un jeu de données complet, comprenant des tables, des contraintes et des relations entre les tables. Étant donné que le **DataSet** est indépendant de la source de données, il 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**.

Le DataAdapter

Chaque fournisseur de données .NET Framework fourni avec le .NET Framework dispose d'un objet **DataAdapter** : le fournisseur de données .NET Framework pour OLE DB inclut un objet **OleDbDataAdapter**, **le fournisseur de données** .NET Framework pour SQL Server inclut un objet **SqlDataAdapter et le fournisseur de données** .NET Framework pour ODBC inclut un objet OdbcDataAdapter. Un **DataAdapter** est utilisé pour extraire les données d'une source de données et remplir les tables dans un **DataSet**. Le **DataAdapter** répercute aussi les modifications apportées au **DataSet** dans la source de données. Le **DataAdapter** utilise l'objet **Connection** du fournisseur de données .NET Framework pour se connecter à une source de données et les objets **Command** pour extraire les données de la source et y répercuter les modifications.

Les objets Command

La propriété **SelectCommand** du **DataAdapter** est un objet **Command** qui extrait les données de la source de données. Les propriétés **InsertCommand**, **UpdateCommand** et **DeleteCommand** du **DataAdapter** sont des objets **Command** qui gèrent les mises à jour dans la source de données conformément aux modifications faites dans le **DataSet**.
La méthode **Fill** du **DataAdapter** est utilisée pour remplir un **DataSet** avec les résultats de **SelectCommand** du **DataAdapter**. **Fill** prend comme arguments un **DataSet** à remplir et un objet **DataTable** ou le nom du **DataTable** à remplir avec les lignes retournées par **SelectCommand**.

L'exemple de code suivant crée une instance d'un **DataAdapter** qui utilise un **Connection** à la base de données **Northwind** Microsoft SQL Server et remplit un **DataTable** dans un **DataSet** avec la liste des clients. L'instruction SQL et les arguments **Connection** passés au constructeur **DataAdapter** sont utilisés pour créer la propriété **SelectCommand** du **DataAdapter**.

```
Dim nwindConn As SqlConnection = New SqlConnection("Data
Source=localhost;Integrated Security=SSPI;Initial Catalog=northwind")

Dim selectCMD As SqlCommand = New SqlCommand("SELECT CustomerID,
CompanyName FROM Customers", nwindConn)
selectCMD.CommandTimeout = 30

Dim custDA As SqlDataAdapter = New SqlDataAdapter
custDA.SelectCommand = selectCMD

nwindConn.Open()

Dim custDS As DataSet = New DataSet
custDA.Fill(custDS, "Customers")

nwindConn.Close()
```

Notez que le code n'ouvre pas et ne ferme pas la **Connection** de manière explicite. La méthode **Fill** ouvre implicitement la **Connection** que le **DataAdapter** utilise si la connexion n'est pas déjà ouverte. Si **Fill** a ouvert la connexion, il la fermera aussi lorsque **Fill** est terminé. Ceci peut simplifier votre code lorsque vous ne traitez qu'une seule opération telle que **Fill** ou **Update**. Cependant, si vous effectuez plusieurs opérations qui nécessitent une connexion ouverte, vous pouvez améliorer la performance de votre application en appelant de manière explicite la méthode **Open** de **Connection**, en effectuant les opérations sur la source de données puis en appelant la méthode **Close** de **Connection**. Vous devez chercher à réduire le temps d'ouverture des connexions à la source de données afin de libérer la ressource utilisée par les autres applications clientes.

Chapitre 5 : MISE A JOUR DE LA BASE DE DONNEES AVEC DATAADAPTER ET DATASET

La méthode Update

La méthode **Update** du **DataAdapter** est appelée pour répercuter les modifications provenant d'un **DataSet** dans la source de données.

Lorsque vous appelez la méthode **Update**, le **DataAdapter** analyse les modifications apportées et exécute la commande appropriée (INSERT, UPDATE ou DELETE). Lorsque le **DataAdapter** trouve une modification dans un **DataRow**, il utilise **InsertCommand**, **UpdateCommand ou DeleteCommand** pour traiter la modification.

Pour gérer les exceptions qui peuvent se produire pendant un **Update**, utilisez l'événement **RowUpdated** pour répondre aux erreurs de mise à jour des lignes. Vous pouvez aussi affecter **true** à **DataAdapter.ContinueUpdateOnError** avant d'appeler **Update** et répondre aux informations d'erreur stockées dans la propriété **RowError** d'une ligne particulière lorsque **Update** est terminé.

Les exemples suivants illustrent l'exécution des mises à jour des lignes modifiées en définissant de manière explicite le **UpdateCommand** du **DataAdapter**. Notez que le paramètre spécifié dans la clause WHERE de l'instruction UPDATE est défini pour utiliser la valeur **Original** de **SourceColumn**. Ceci est important, car la valeur **Current** peut avoir été modifiée et ne pas correspondre à la valeur dans la source de données. La valeur **Original** est la valeur qui a été utilisée pour remplir le **DataTable** à partir de la source de données.

```
Dim catDA As SqlDataAdapter = New SqlDataAdapter("SELECT CategoryID, CategoryName FROM
Categories", nwindConn)

catDA.UpdateCommand = New SqlCommand("UPDATE Categories SET CategoryName = @CategoryName " & _
                                     "WHERE CategoryID = @CategoryID", nwindConn)

catDA.UpdateCommand.Parameters.Add("@CategoryName", SqlDbType.NVarChar, 15, "CategoryName")

Dim workParm As SqlParameter = catDA.UpdateCommand.Parameters.Add("@CategoryID",
SqlDbType.Int)
workParm.SourceColumn = "CategoryID"
workParm.SourceVersion = DataRowVersion.Original

Dim catDS As DataSet = New DataSet
catDA.Fill(catDS, "Categories")

Dim cRow As DataRow = catDS.Tables("Categories").Rows(0)
cRow("CategoryName") = "New Category"

catDA.Update(catDS)
```

Ordre des insertions, mises à jour et suppressions

Dans de nombreuses circonstances, l'ordre dans lequel les modifications faites dans le **DataSet** sont transmises à la source de données est important. Par exemple, si une valeur de clé primaire d'une ligne existante est mise à jour et qu'une nouvelle ligne est ajoutée avec la nouvelle valeur de clé primaire, il est important de traiter la mise à jour avant l'insertion.
Vous pouvez utiliser la méthode **Select** du **DataTable** pour retourner un tableau **DataRow** qui fait uniquement référence à des lignes avec un **RowState** particulier. Vous pouvez alors passer le tableau **DataRow** retourné à la méthode **Update** du **DataAdapter** pour traiter les lignes modifiées. En spécifiant un sous-ensemble des lignes à mettre à jour, vous pouvez contrôler l'ordre dans lequel les insertions, mises à jour et suppressions sont traitées.
Le code suivant garantit, par exemple, que seront d'abord traitées les lignes supprimées de la table puis les lignes mises à jour et enfin les lignes insérées.

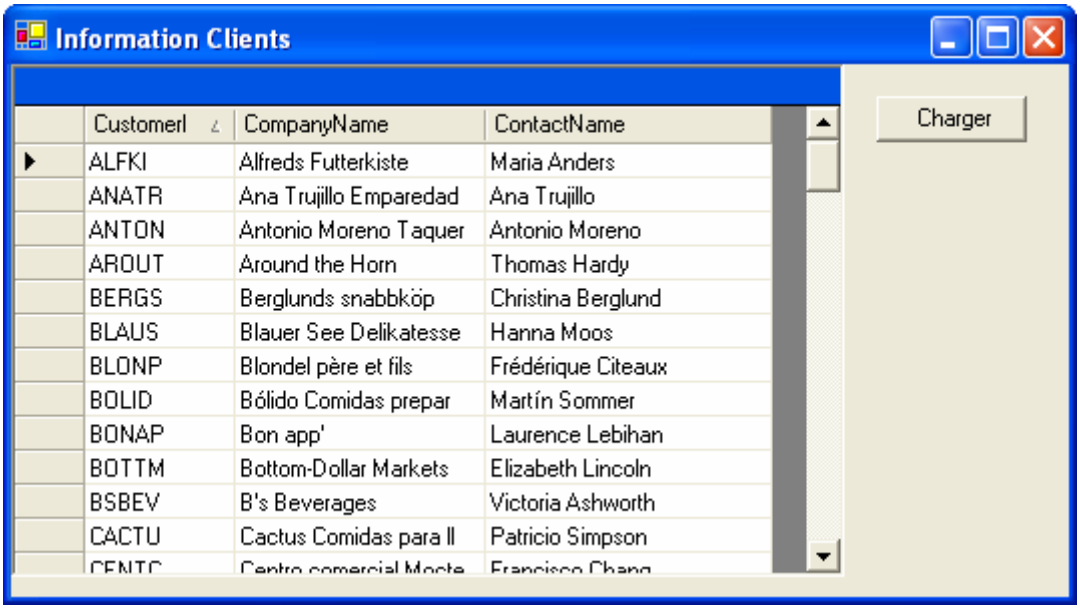
```
Dim updTable As DataTable = custDS.Tables("Customers")

' First process deletes.
custDA.Update(updTable.Select(Nothing, Nothing, DataViewRowState.Deleted))

' Next process updates.
custDA.Update(updTable.Select(Nothing, Nothing, DataViewRowState.ModifiedCurrent))

' Finally, process inserts.
custDA.Update(updTable.Select(Nothing, Nothing, DataViewRowState.Added))
```

Chapitre 6 : LIAISON DES DONNEES AVEC WINDOWS FORMS

Création d'un formulaire Windows simple dépendant

L'utilisation la plus simple de la liaison de données dans l'environnement .NET sert à afficher le contenu d'une table dans une grille. Pour l'exemple ci-dessous, suivez les étapes ci-après :

1. Créez un formulaire Windows.
2. Créer et configurer le groupe de données auquel vous voulez lier le formulaire.
3. Ajoutez un contrôle de grille de données au formulaire et liez-le aux données.



Création de l'exemple de formulaire

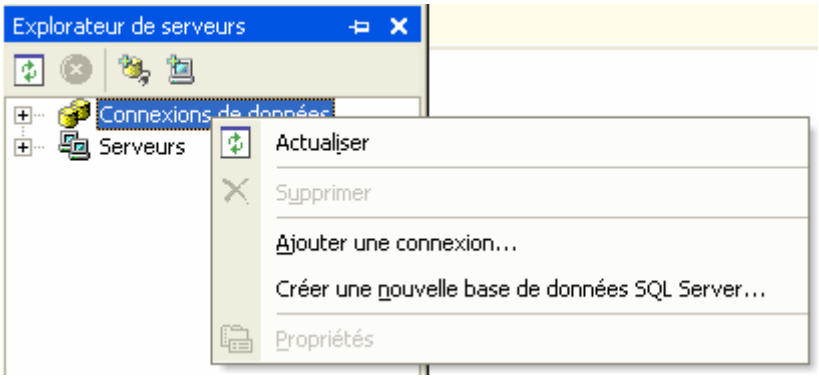
Vous trouverez ci-dessous les étapes détaillées de la création d'un exemple de formulaire.

1. Cliquez sur **Fichier**, **Nouveau**, puis **Projet**. La boîte de dialogue **Nouveau projet** s'affiche.

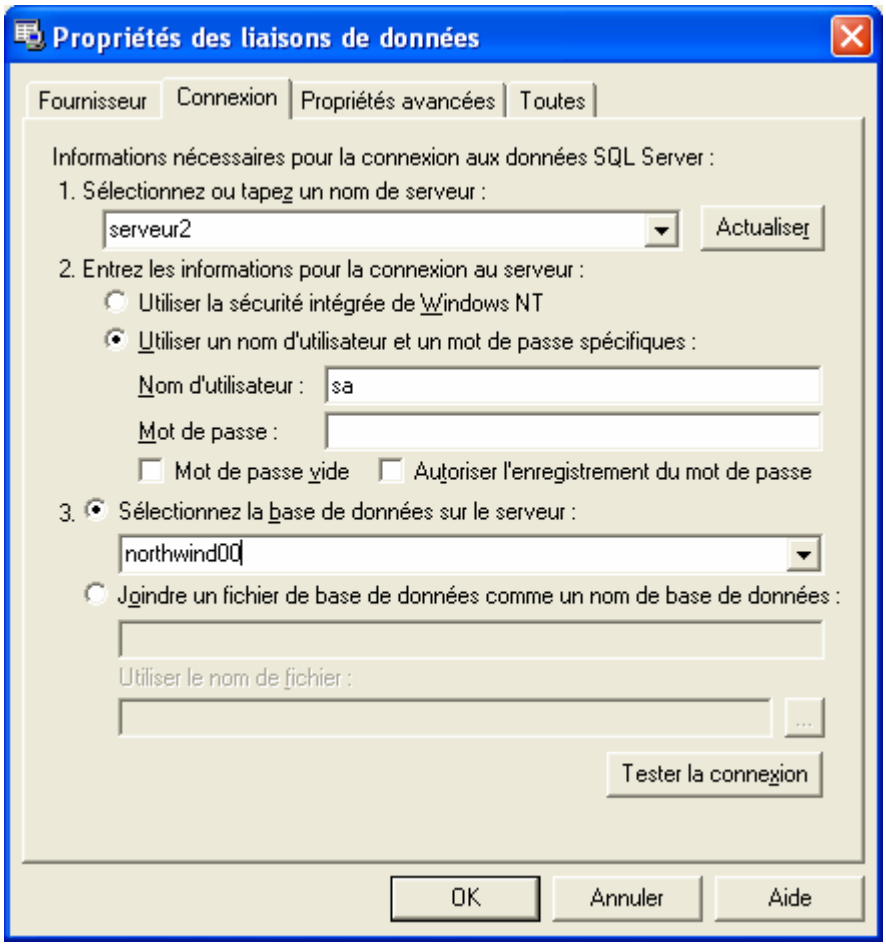
2. Sélectionnez le **Type de projet** dans l'arborescence située à gauche de la boîte de dialogue. Dans cet exemple, sélectionnez **Projets Visual Basic**.
3. Sélectionnez Application Windows dans la liste des modèles située à droite de la boîte de dialogue.
4. Indiquez le nom et l'emplacement du projet que vous voulez créer. Cliquez sur **OK** pour créer le projet.
5. Appuyez sur **F4** pour afficher les propriétés du formulaire. Changez la propriété **Name** du formulaire pour **frmClients**.
6. Changez la propriété **Text** pour **Informations Clients**.

Création d'une nouvelle connexion

Dans l'explorateur de serveurs, cliquez avec le bouton droit de la souris sur « Connexions de données », sélectionnez ensuite l'option ajouter une nouvelle connexion



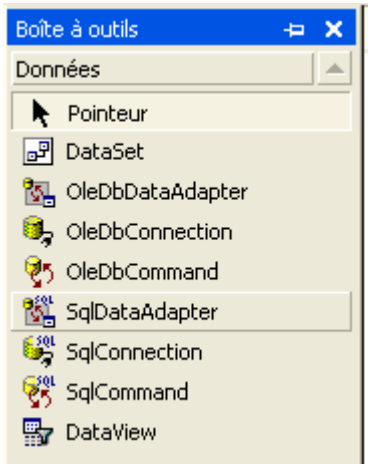
1. la boite de dialogue **Propriétés des liaisons de données** s'affiche.



2. Changez le nom du serveur pour qu'il pointe vers celui de votre serveur SQL.
3. Changez le nom et le mot de passe de l'utilisateur nécessaire pour se connecter à ce serveur SQL.
4. Sélectionnez la base de données **Northwind** qui sera utilisée dans l'exemple.

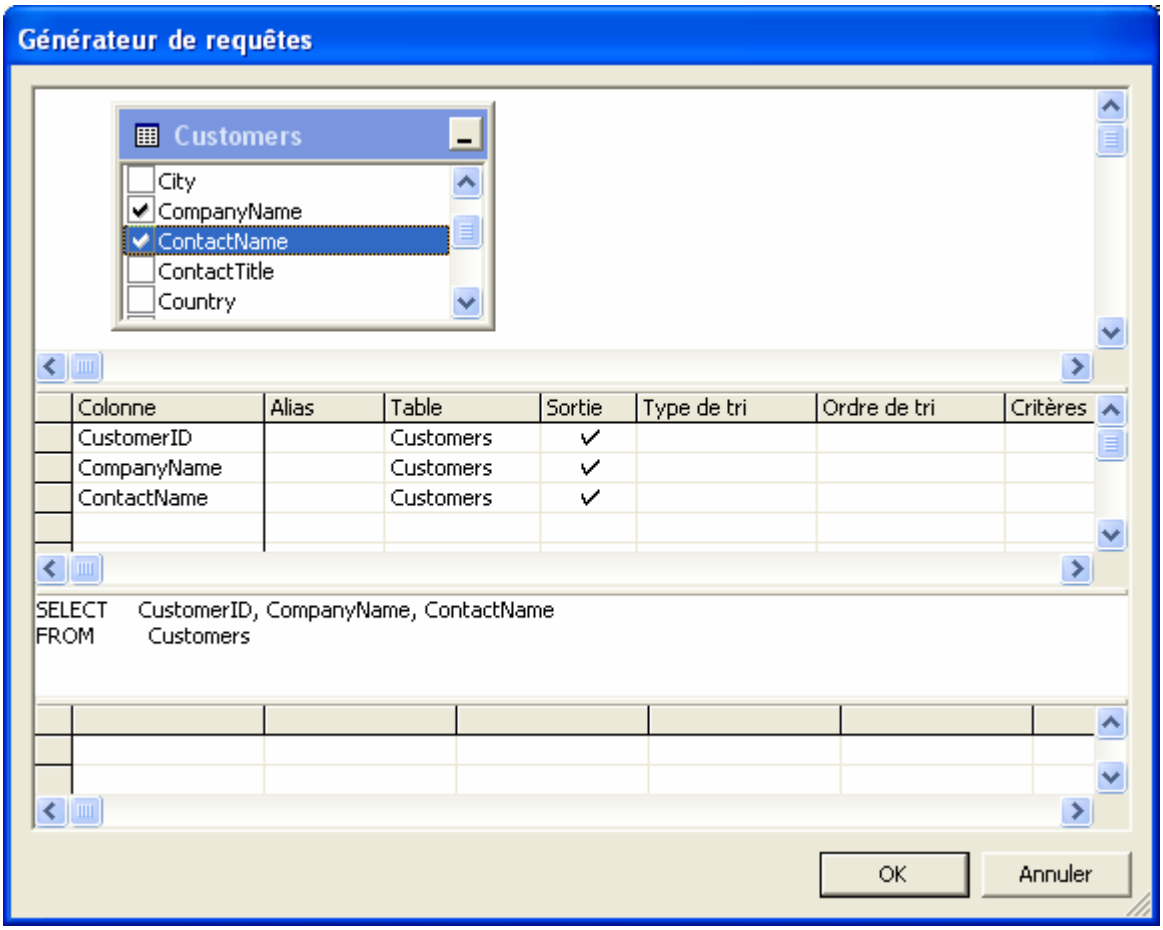
Création d'un adaptateur de données

1. Dans la boîte à outils, dans l'onglet **Données**, sélectionnez un SQLDataAdapter. Faites-le glisser et déposez-le sur le formulaire.



2. L'Assistant Configuration d'adaptateur de données s'affiche. Cliquez sur **Suivant**.
3. L'assistant vous permet de désigner le type de requête. Sélectionnez **Utiliser des instructions SQL** et cliquez sur **Suivant**. Tapez une instruction SQL et cliquez sur **Générateur de requête** pour que l'assistant crée l'instruction SQL à votre place. Dans cet exemple, tapez l'instruction SQL suivante :

```
SELECT CustomerID, CompanyName, ContactName, ContactTitle,
Country FROM Customers
```



4. Cliquez sur **Suivant** puis sur **Terminer** pour effectuer le processus. Notez qu'un objet SQLConnection nommé **SQLConnection1** et qu'un objet SQLDataAdapter nommé **SQLDataAdapter1** apparaissent dans la barre d'état du formulaire.

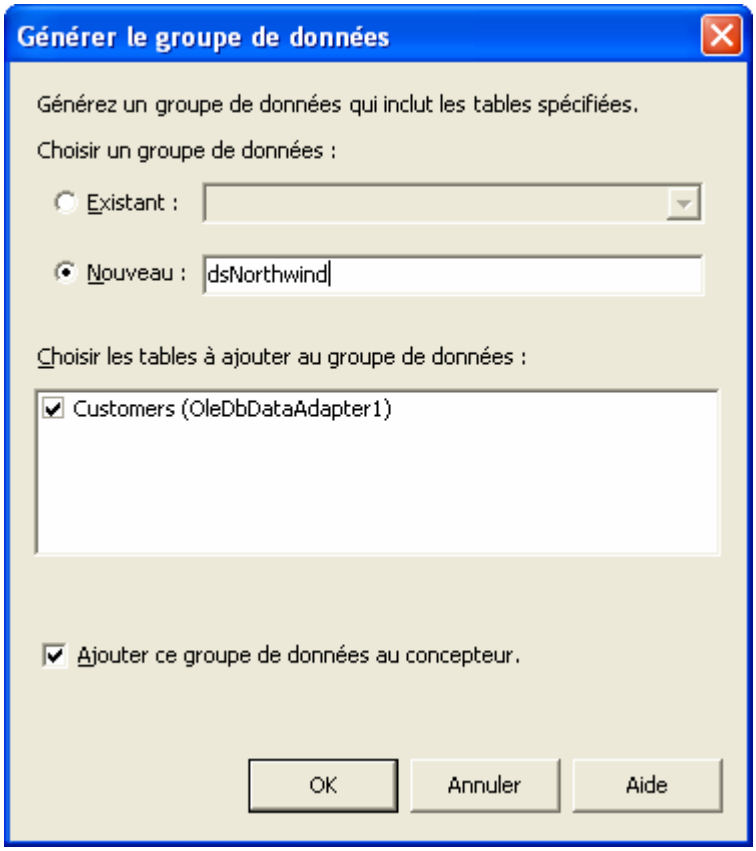
L'objet **SQLConnection1** contient des informations concernant l'accès à la base de données sélectionnée. L'objet **SQLDataAdapter1** contient une requête définissant les tables et les colonnes de la base de données à laquelle vous voulez accéder.

Remarque Dans notre exemple, le SQLDataAdapter est sélectionné depuis la boîte à outils car vous utilisez un serveur SQL. Cet objet ne vous permet de vous connecter qu'à des bases de données du serveur SQL. Si vous souhaitez vous connecter à d'autres types de données OLEDB, sélectionnez l'OLEDBDataAdapter plus générique.

Création de la classe de groupe de données

L'étape suivante consiste à créer une classe de groupe de données basée sur la requête sélectionnée lors de la création de l'adaptateur de données. Pour créer la classe de groupe de données :

1. Cliquez avec le bouton droit sur l'objet **SQLDataAdapter1**, puis sur **Générer le groupe de données**. La boîte de dialogue **Générer le groupe de données** s'affiche.



2. Entrez **dsNorthwind** comme nom du nouveau groupe de données que vous créez. Assurez-vous d'avoir activé la case à cocher **Ajouter ce groupe de données au concepteur.** Cliquez sur **OK**. Ceci crée le groupe de données.

Lorsque cette étape est terminée, vous voyez un nouveau contrôle, dsNorthwind1, dans la barre d'état de ce formulaire. Ce nouveau contrôle est une référence au fichier dsNorthwind.xsd qui a également été ajouté à votre fenêtre Explorateur de solutions. dsNorthwind.xsd est une définition de schéma XML de l'instruction SQL que vous avez tapée précédemment. Il existe une classe derrière ce fichier XSD que vous ne pouvez voir sauf si vous cliquez sur **Projet**, puis sur **Afficher tous les fichiers** dans le menu. Vous pouvez ensuite cliquer sur le signe plus qui s'affiche derrière le fichier dsNorthwind.xsd pour voir le fichier dsNorthwind.vb.

La classe dsNorthwind.vb a des propriétés qui correspondent au groupe de données courant et des propriétés qui correspondent à chaque colonne spécifiée dans votre instruction SQL. Bien que cette classe ne vous soit d'aucune utilité, vous devez savoir qu'elle se trouve là.

Ajout d'un contrôle de grille de données pour afficher les données

À présent, vous êtes prêt à créer un formulaire pour afficher les données contenues dans le groupe de données. Dans cet exemple, vous ajouterez un contrôle de grille de données unique au formulaire. Le contrôle de grille de données affichera les données contenues dans le groupe de données. Voici ce que vous devez faire :

1. En haut de la fenêtre, cliquez sur l'onglet pour afficher le formulaire frmCustomers.
2. Dans l'onglet **Windows Forms** de la boîte de dialogue, faites glisser un contrôle de grille de données sur le formulaire. Dimensionnez le contrôle selon vos besoins.
3. Appuyez sur **F4** pour afficher les propriétés du contrôle.
4. Pour la propriété **DataSource**, sélectionnez dsNorthwind1.
5. Pour la propriété **DataMember**, sélectionnez Customers.

Les étapes que vous venez de terminer ont lié le groupe de données au contrôle de grille de données. Il suffit d'une étape supplémentaire pour voir les données s'afficher dans le contrôle de grille de données.

Remplissage du contrôle de grille de données avec des données

Bien que le contrôle de grille de données soit lié au groupe de données, ce dernier n'est pas rempli. Nous allons ajouter un boutons de commande qui permettra de remplir la grille.

Faites glisser un bouton de commande su le formulaire

Pour la propriété **name,** tapez cmdCharger

Pour la propriété **text,** tapez Charger

Double cliquez sur le bouton et tapez le code suivant

```
  SqlDataAdapter1.Fill(dsNorthwind1, "Customers")
```

À présent, vous êtes prêt à voir les données de la table Customers affichées dans le contrôle de grille de données. Il reste une étape avant que vous ne puissiez exécuter ce formulaire. Du fait que vous avez modifié le nom du formulaire, vous devez indiquer au projet quel sera le formulaire de démarrage.

1. Dans la fenêtre Explorateur de solutions, cliquez sur le nom de votre projet.
2. Cliquez sur le projet avec le bouton droit et choisissez **Propriétés** dans le menu contextuel.
3. Cliquez sur la liste modifiable **Objet de démarrage** et choisissez **frmCustomers** dans la liste.
4. Cliquez sur **OK**.
5. Appuyez sur **F5** pour exécuter ce projet et, si vous avez effectué toutes les opérations correctement, vous devez à présent voir s'afficher les données des clients dans le contrôle de grille de données après click sur le bouton **Chager**.

Utilisation des listes modifiables et des zones de listes

L'exemple que vous venez de créer convient bien pour les petits groupes de données, mais sera peu efficace si la table customers contient des milliers d'enregistrements. Le problème est qu'il récupère tous les clients de la table Customers et les affiche dans le contrôle de grille de données. Bien que cette technique soit très bonne lorsqu'il s'agit d'une démonstration, elle se révèle inadaptée à une application de production.

Un serveur de base de données est optimisé pour traiter des volumes importants de données, mais il est souhaitable de ne récupérer que des ensembles de résultats de petite taille. L'exemple de la section précédente serait plus efficace si l'utilisateur commençait par limiter les données en sélectionnant un pays particulier dans la liste modifiable, puis n'affiche que les clients de ce pays. Cette section vous montre comment créer la liste modifiable dépendante. La section suivante vous montre comment

utiliser la liste modifiable pour limiter les données affichées dans le contrôle de grille de données. Cet exemple nécessite les étapes de base suivantes :

1. Ajout d'un second adaptateur de données au formulaire.
2. Création d'un second groupe de données.
3. Ajout d'une liste modifiable au formulaire.
4. Liaison de la liste modifiable au second adaptateur de données.
5. Remplissage de la liste modifiable avec les données du second adaptateur de données.

Ajout d'un second adaptateur de données au formulaire

Avant d'ajouter une liste modifiable au formulaire, commencez par ajouter un second adaptateur de données au formulaire. Vous utiliserez cet adaptateur de données pour créer un groupe de données contenant une liste unique de pays à partir de la table Customers. Pour ajouter le second adaptateur de données :

1. Dans la boîte à outils, sous l'onglet **Données**, sélectionnez un SQLDataAdapter. Faites-le glisser et déposez-le sur le formulaire. L'Assistant Configuration d'adaptateur de données s'affiche. Cliquez sur **Suivant**.
2. Sélectionnez la connexion à la base de données Northwind que vous avez créée dans l'exemple précédent. Cliquez sur **Suivant**.
3. Sélectionnez **Utiliser des instructions SQL** et cliquez sur **Suivant**.
4. Entrez l'instruction SQL suivante :

```
SELECT DISTINCT Country FROM Customers
```

5. Cliquez sur le bouton **Options avancées** de cette boîte de dialogue.
6. N'activez pas la case à cocher Générer des instructions Insert, Update et Delete comme dans la figure 6. Ces instructions ne sont pas nécessaires pour un groupe de données utilisé uniquement pour remplir une liste modifiable. Cliquez sur **OK** pour fermer cette boîte de dialogue.
7. Cliquez sur **Terminer** pour générer le SQLDataAdapter.

Un objet nommé **SQLDataAdapter2** sera désormais ajouté à la barre d'état de votre formulaire.

Génération du groupe de données

1. Cliquez avec le bouton droit sur objet nommé **SQLDataAdapter2**, puis sur **Générer le groupe de données**. La boîte de dialogue **Générer le groupe de données** s'affiche.
2. Cliquez sur **Nouveau** et entrez le nom **dsCountries**. Assurez-vous d'avoir activé la case à cocher **Ajouter ce groupe de données au concepteur.** Cliquez sur **OK**. Ceci crée le groupe de données.

Vous avez ajouté un nouvel objet nommé dsCountries1 à la barre d'état et un nouveau fichier XSD nommé dsCountries.xsd à l'Explorateur de solutions. Ces deux éléments créent ensemble la classe de groupe de données et servent à effectuer la liaison de données sur la liste modifiable que vous allez créer.

Ajout d'une liste modifiable au formulaire

À présent, vous êtes prêt à ajouter la liste modifiable au formulaire. Vous devez agrandir le formulaire et déplacer le contrôle de grille de données vers le bas pour faire de la place à cette nouvelle liste modifiable. Après cela, dans la boîte à outils, vous pouvez faire glisser et déposer une liste modifiable sur le formulaire à partir de l'onglet **Windows Forms**. Placez-la au-dessus du contrôle de grille de donnée.

Liaison de la liste modifiable à l'adaptateur de données

La liste modifiable que vous avez ajoutée n'est liée à aucune donnée. Pour lier la liste modifiable au groupe de données dsCountries :

1. Sélectionnez la liste modifiable.
2. Changez le nom pour **cboCountry**.
3. Définissez la propriété **DataSource** sur dsCountries1. Ceci spécifie la source de données à partir de laquelle ce contrôle de grille de données sera rempli.
4. Changez la propriété **DropDownStyle** pour **DropDownList**.
5. Définissez la propriété **DisplayMember** sur **Customers.Country**. Vous devez cliquer sur la liste déroulante de la propriété **DisplayMember**, puis développer Customers pour afficher la liste des colonnes que vous pouvez utiliser. Cette étape désigne la colonne de la source de données utilisée pour remplir le contrôle de grille de données.

6. Définissez la propriété **ValueMember** du contrôle de grille de données sur **Customers.Country**. La propriété **ValueMember** sert à désigner la valeur courante lorsqu'un élément est sélectionné.

Remplissage de la liste modifiable avec des données

Comme pour le contrôle de grille de données, vous devez écrire le code permettant de remplir la liste modifiable de données. Ce code est ajouté à la procédure d'événement Load.

```
Private Sub frmCustomers_Load

  SqlDataAdapter2.Fill(DsCountries1, "Customers")

End Sub
```

Affichage des données en fonction d'une requête paramétrée

À présent que la liste modifiable est chargée avec les pays, il est temps d'utiliser le pays sélectionné dans la liste modifiable pour sélectionner les clients à charger dans le contrôle de grille de données. Les étapes principales sont les suivantes :

1. Modifiez la propriété **SelectCommand** de l'objet **SqlDataAdapter1** pour qu'il accepte un paramètre du pays pour lequel vous voulez afficher des données.
2. Ajoutez le code qui s'exécute lorsque l'utilisateur sélectionne un élément dans la liste modifiable.

Modifiez la propriété SelectCommand

L'étape suivante consiste à modifier la propriété **SelectCommand** de l'adaptateur de données de manière à ce qu'il soit basé sur la requête paramétrée. Les étapes sont les suivantes :

1. Dans la barre d'état du formulaire, cliquez sur l'objet **SqlDataAdapter1**.
2. Appuyez sur **F4** pour ouvrir la fenêtre **Propriétés**.
3. À côté de la propriété **SelectCommand**, cliquez sur le signe plus pour développer la liste des sous-propriétés.
4. Cliquez sur la propriété **CommandText**. Cliquez sur **Générer (...)** pour afficher la boîte de dialogue **Générateur de requête**.
5. Ajoutez une clause WHERE afin que l'instruction SQL ressemble à ceci :

```
SELECT CustomerID, CompanyName, ContactName, Country
```

```
FROM Customers
WHERE Country = @CountryParam
```

6. Cliquez sur **OK** pour accepter cette modification.
7. Cliquez sur l'objet **SQLDataAdapter1**.
8. Cliquez sur **Données**, sur **Générer le groupe de données**, puis sur **OK** pour régénérer le groupe de données existant.

Ajoutez le code qui s'exécute lorsque l'utilisateur sélectionne un élément dans la liste modifiable

La partie finale de code remplit le contrôle de grille de données lorsqu'un pays est sélectionné dans la liste modifiable **cboCountry**. Les étapes principales sont les suivantes :

1. Répondez à l'événement **SelectedIndexChanged**.
2. Définissez le paramètre dans le groupe de données avec les données récupérées à partir de l'élément sélectionné dans la liste modifiable.
3. Remplissez le groupe de données à l'aide de ce paramètre.

Suivez ces étapes pour ajouter le code et permettre que cela fonctionne.

1. Ouvrez le formulaire en mode création.
2. Double-cliquez sur la liste modifiable **cboCountry** pour afficher la procédure d'événement **SelectedIndexChanged**. Il s'agit de l'événement déclenché lorsque vous choisissez un nouvel élément dans la liste modifiable.
3. Écrivez le code suivant dans cette procédure.

```
Private Sub cboCountry_SelectedIndexChanged

  ' Obtention de l'objet Parameter et définition de sa valeur
  SqlDataAdapter1.SelectCommand.Parameters.Item("@CountryParam").Value = _
     cboCountry.SelectedValue

  ' Suppression du groupe de données
  DsNorthwind1.customers.rows.Clear()
  ' Chargement du groupe de données en utilisant la valeur du paramètre
  SqlDataAdapter1.Fill(DsNorthwind1, "Cutomers")
End Sub
```

Vous devez utiliser l'objet **SelectCommand** de l'adaptateur de données pour récupérer l'objet **Parameter** spécifié. Vous pouvez ensuite définir la propriété **Value** de ce paramètre sur la propriété **SelectedValue** de la liste modifiable. La propriété **SelectedValue** est remplie avec le nom du pays car vous avez défini la propriété **ValueMember** de la liste modifiable pour qu'elle utilise le champ **Country** du groupe de données. Après avoir rempli cette valeur, vous pouvez remplir le groupe de données du client et

le contrôle de grille de données sera rempli automatiquement par les seuls clients du pays sélectionné.

Formulaire de mise à jour d'une table

Les exemples précédents ont tous affiché les données dans un contrôle de grille de données. Lorsque vous créez des applications, il est également nécessaire d'afficher des colonnes individuelles dans des zones de texte d'un formulaire pour les éditer. Bien que le propos de ce document ne soit pas de vous faire apprendre l'édition des données, vous allez apprendre à afficher des données dans les zones de texte. Les étapes principales sont les suivantes :

1. Créer un formulaire Windows.
2. Créer et configurer le groupe de données auquel vous voulez lier le formulaire.
3. Ajouter des contrôles au formulaire et liez-les au groupe de données.
4. Ajouter des boutons de commande permettant à l'utilisateur de naviguer d'une ligne à l'autre.

Ajout de contrôles au formulaire et liaison de ces contrôles au groupe de données

Ajoutez des contrôles au formulaire et définissez leurs propriétés comme dans le tableau 1.

Tableau des contrôles utilisés pour créer le formulaire

Type de contrôle	Propriété	Valeur
Label	Name	Label1
	Text	Code
TextBox	Name	txtCustomerID
Label	Name	Label2
	Text	Nom
TextBox	Name	txtCompanyName
Label	Name	Label3
	Text	Contact
TextBox	Name	txtContactName
CommandButton	Name	cmdPremier
	Text	<<
CommandButton	Name	cmdPrecedent
	Text	<
CommandButton	Name	cmdSuivant
	Text	>
CommandButton	Name	cmdDernier
	Text	>>
CommandButton	Name	cmdCharger
	Text	Charger
CommandButton	Name	cmdAjouter
	Text	Ajouter
CommandButton	Name	cmdSupprimer
	Text	Supprimer
CommandButton	Name	cmdMAJ
	Text	Mettre à jour

Vous devez à présent lier chaque zone de texte à une colonne du groupe de données. Pour ce faire :

1. Sélectionnez la zone de texte que vous voulez lier.
2. Appuyez sur **F4** pour afficher la fenêtre **Propriétés**.
3. Cliquez pour développer les propriétés **DataBindings**.
4. Sous **DataBindings**, sélectionnez la propriété **Text**.
5. Ouvrez la liste modifiable et liez chaque zone de texte au champ approprié. Par exemple, pour lier la zone de texte **txtCustomerID** au champ **CustomerID**, cliquez sur le **signe plus** de

DsCustomers1, sur le **signe plus** de Customers, et sélectionnez le champ **CustomerID**.

Ecrivez le code permettant de remplir le formulaire de données. Pour cela double cliquez sur le bouton charger puis tapez le code :

```
    DsNorthwind1.customers.Clear()
    SqlDataAdapter1.Fill(DsCustomers1, "Clients")
  End Sub
```

L'objet CurrencyManager

Assure un suivi de la position et supervise les liaisons à une source de données. Il existe, dans le formulaire, un objet CurrencyManager pour chaque source de données discrète à laquelle vous vous liez. Si les contrôles du formulaire sont tous liés à la même source (par exemple, si plusieurs contrôles TextBox sont liés à la même table de données), ils partagent alors le même CurrencyManager

Ajout de boutons de commande permettant à l'utilisateur de naviguer d'une ligne à l'autre

Commençons par déclarer une donnée membre de type CurrencyManager dans notre formulaire que nous appellerons cmCustomers.

Private cmCustomers as CurrencyManager

Nous devons ensuite récupérer le CurrencyManager au chargement du formulaire à partir de la collection BindingContext.

```
Private Sub Form1_Load
        cmCustomers = BindingContext(dsNorthwind1, "Customers")
End Sub
```

Code des boutons de navigation

```
Private Sub cmdPrecedent_Click
    cmCustomers Position -= 1
End Sub

Private Sub cmdPremier_Click
    cmCustomers.Position = 0
End Sub

Private Sub cmdDernier_Click
    cmCustomers.Position = dsNorthwind1.Customers.Rows.Count - 1
End Sub
```

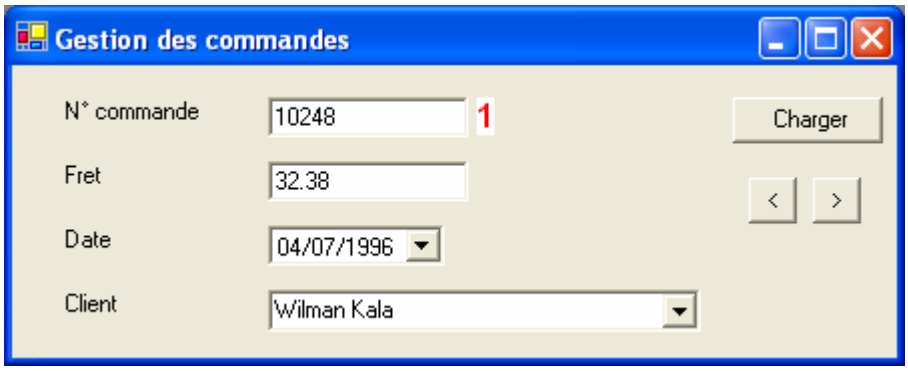
Ajout de boutons de commande de mise à jour

Le code suivant permet l'ajout d'un nouvel enregistrement, la suppression d'un enregistrement existant et enfin la mise à jour de la base de données

```
Private Sub cmdAjouter_Click
    cmCustomers.AddNew()
End Sub

Private Sub cmdSupprimer_Click
    cmCustomers.RemoveAt(cmCustomers.Position)
End Sub

Private Sub cmdMAJ_Click
    daCustomers.Update(dsNorthwind1)
End Sub
```

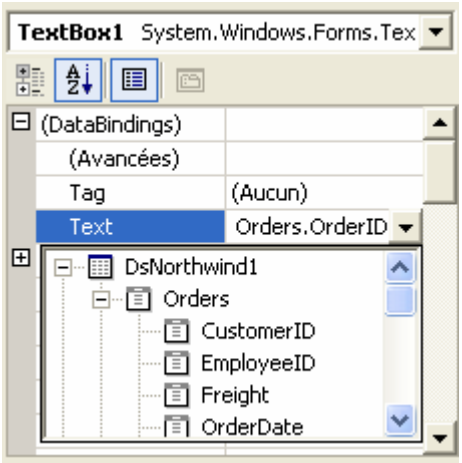
Chapitre 7 : Quelques contrôles dépendants

Zone de texte, Label



Pour lier une zone de texte à une source de données :

En mode design, utiliser la propriété text du DataBinding.



Dans l'exemple ci-dessus, la zone de texte TextBox1 est liée a champs OrderID de la table Orders du dataSet DsNorthwind1.

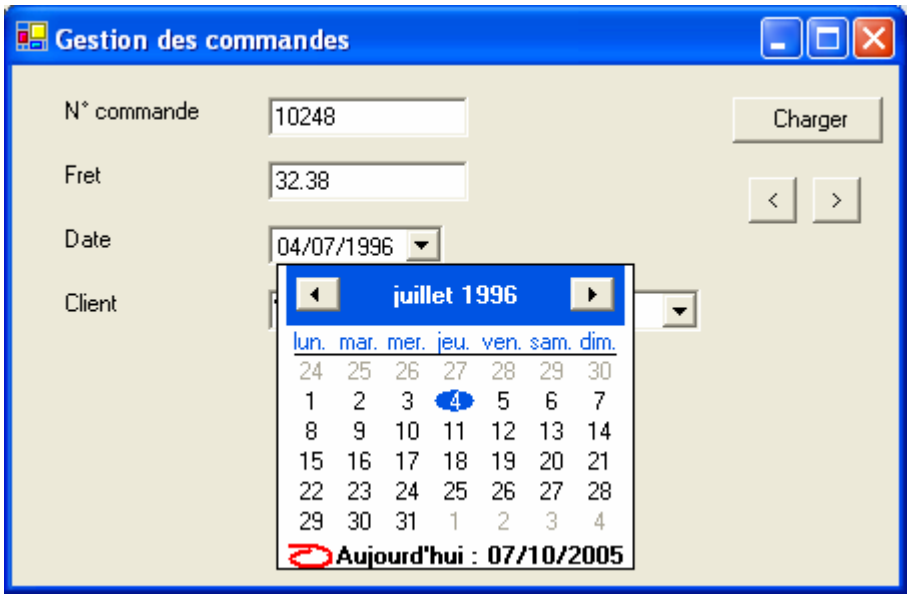
Cette liaison peut être effectuée au cours de l'exécution comme suit :

```
TextBox1.DataBindings.Add(New System.Windows.Forms.Binding("Text",
DsNorthwind1, "Orders.OrderID"))
```

Un label peut être lié de la même manière qu'une zone de texte, seulement il ne peut être utilisé que pour afficher des données.

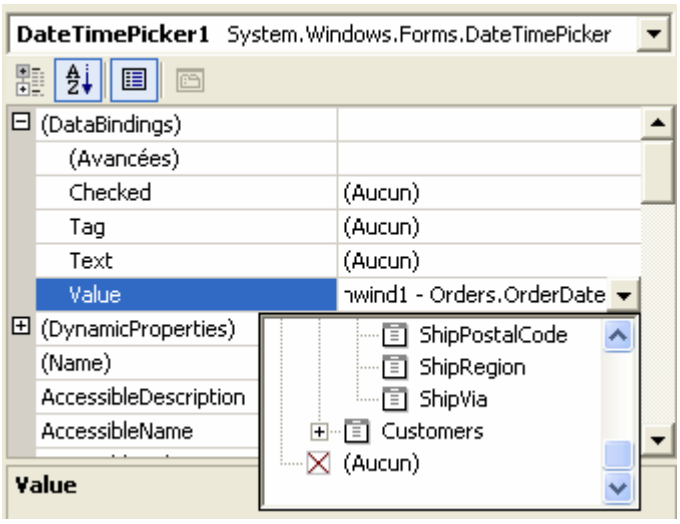
DateTimePeacker

Ce contrôle est utilisé pour permettre à l'utilisateur de sélectionner une date et/ou une heure, et pour afficher cette valeur date et heure au format spécifié. Vous pouvez restreindre la fourchette de dates et d'heures qu'il sera possible de sélectionner en définissant les propriétés MinDate et MaxDate.



La propriété **Format** détermine le format du contrôle qui peut être Long, Short, Time ou Custom. Si la propriété **Format** a pour valeur Custom, vous pouvez créer votre propre style de format en définissant la propriété **CustomFormat.**
Afin d'utiliser un contrôle de style up-down pour sélectionner la valeur de date-heure, affectez à la propriété **ShowUpDown** la valeur **true**. Lors de sa sélection, le contrôle Calendar ne se déroulera pas. La date et l'heure peuvent être réglées par la sélection de chaque élément individuellement et par l'utilisation des boutons haut et bas pour en modifier la valeur.

Pour lier ce contrôle :
En mode création, utilisez la propriété Value du DataBinding.



Cet exemple permet de lier le contrôle DateTimePeaker1 au champ OrderDate de la table Orders.

Au cours de l'exécution, tapez le code suivant :

```
DateTimePicker1.DataBindings.Add(New System.Windows.Forms.Binding("Value",
DsNorthwind1, "Orders.OrderDate"))
```

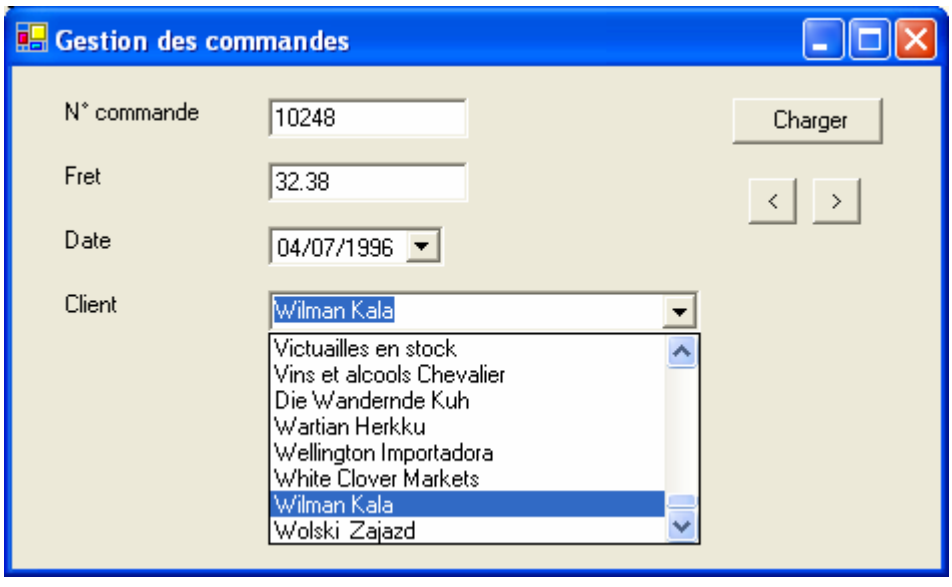
ComboBox, ListBox

Un **ComboBox** affiche un champ d'édition associé à un **ListBox**, permettant à l'utilisateur de sélectionner dans la liste ou d'entrer un nouveau texte. La propriété **DropDownStyle** détermine le style de la zone de liste déroulante à afficher. Pour afficher toujours une liste que l'utilisateur ne peut pas modifier, utilisez un contrôle **ListBox**.

En plus de l'affichage et de la fonctionnalité de sélection, le **ComboBox** comprend également des fonctionnalités qui vous permettent d'ajouter efficacement des éléments au **ComboBox** et de rechercher du texte dans les éléments de la liste. Les méthodes **BeginUpdate** et **EndUpdate** vous permettent d'ajouter un grand nombre d'éléments au **ComboBox** sans repeindre le contrôle chaque fois qu'un élément est ajouté à la liste. Les méthodes **FindString** et **FindStringExact** vous permettent de rechercher un élément dans la liste qui contient une chaîne recherchée spécifique.

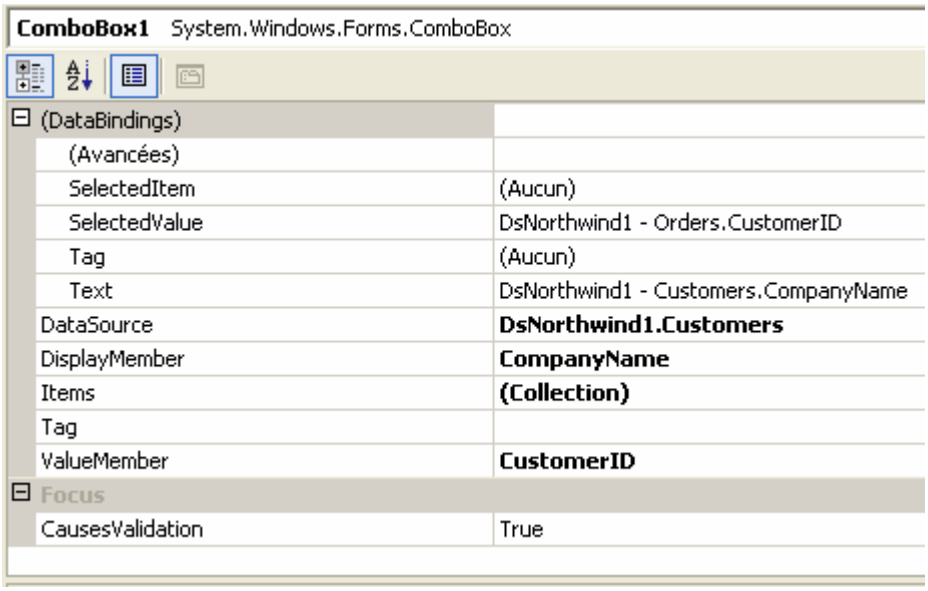
La propriété **Text** spécifie la chaîne affichée dans le champ d'édition.

La propriété **SelectedIndex** renvoie ou définit l'élément en cours.



Pour lier ce contrôle à une source de données :
En mode création utilisez les propriétés suivantes

DataSource : indique la source qui sera utilisée pour ajouter des objets au contrôle.
DisplayMember : spécifie le champ qui sera affiché dans le ComboBox.
ValueMember : Nom de champ qui sera utilisé comme valeur retenue après le choix de l'utilisateur.
SelectedValue du DataBinding : Champ de la table liée au contrôle.
Text du DataBinding : Champ qui sera affiché dans la zone de texte.



Au cours de l'exécution, la liaison peut être faite avec le code suivant :

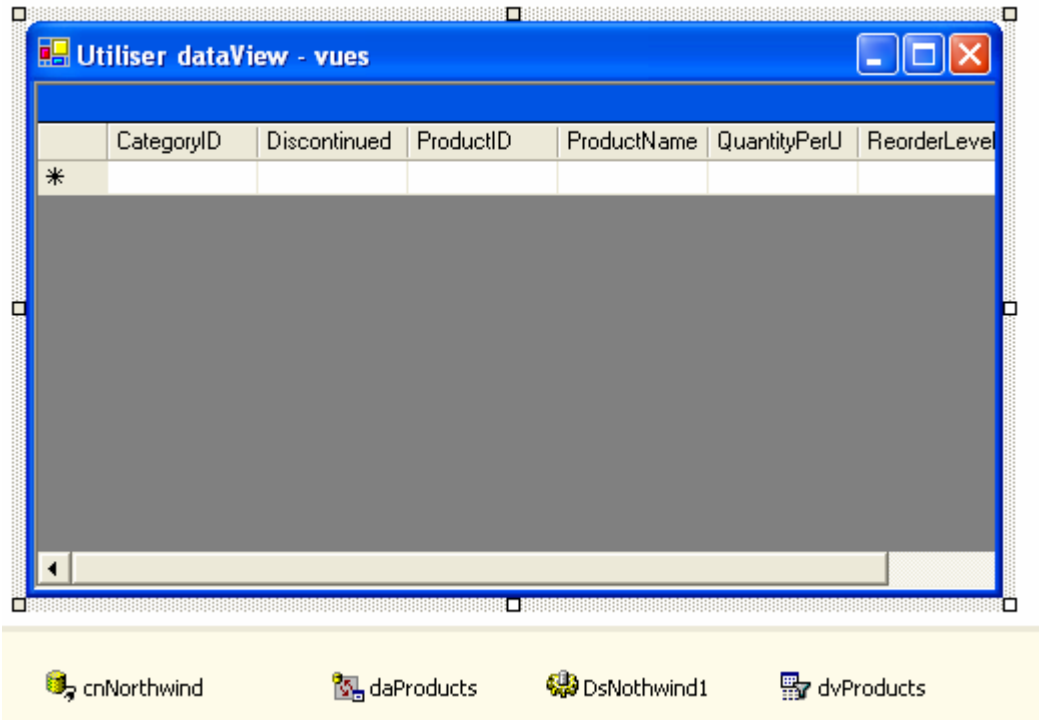
```
ComboBox1.DataBindings.Add(New
System.Windows.Forms.Binding("SelectedValue", DsNorthwind1,
"Orders.CustomerID"))
ComboBox1.DataBindings.Add(New System.Windows.Forms.Binding("Text",
DsNorthwind1, "Customers.CompanyName"))
ComboBox1.DataSource = DsNorthwind1.Customers
ComboBox1.DisplayMember = "CompanyName"
ComboBox1.ValueMember = "CustomerID"
```

Chapitre 8 : Création et utilisation de DataViews

Un **DataView** vous permet de créer différentes vues des données stockées dans un **DataTable**, possibilité qui est souvent utilisée dans les applications de liaison de données. En utilisant un **DataView**, vous pouvez présenter les données d'une table en appliquant différents ordres de tri et filtrer les données en fonction d'un état de ligne ou d'une expression de filtre.

Création d'un DataView

Création d'un dataView en mode conception :



1. Commencez par créer un dataAdapter et un dataset
2. A partir de la boite à outils, glissez un dataView sur votre formulaire et renseignez sa propriété Table.
3. Liez ensuite la grille de données au dataview que vous venez de créer

Création d'un dataView au cours de l'exécution :

Vous pouvez utiliser le constructeur de **DataView** ou créer une référence à la propriété **DefaultView** du **DataTable**.

L'exemple de code suivant crée un **DataView** lié à la table Customers avec un filtre sur les clients des USA et un tri sur le champ contactName.

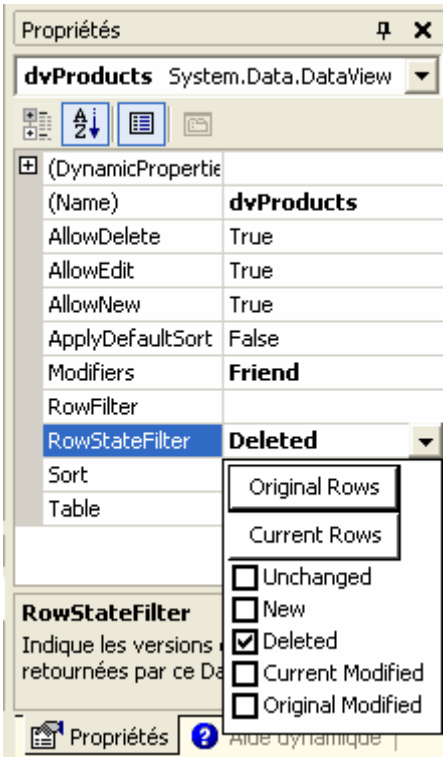
```
Dim dvCustomers As DataView = New DataView(DsNorthwind1.Customers, _
                                           "Country = 'USA'", _
```

```
                                        "ContactName", _
                                        DataViewRowState.CurrentRows)
```

L'exemple de code suivant montre comment obtenir une référence au **DataView** par défaut d'un **DataTable** à l'aide de la propriété **DefaultView** de la table.

```
Dim dvProduits As DataView = dsNorthwind1.Customers.DefaultView
```

Filtrage par état de ligne



À l'aide de la propriété **RowStateFilter**, vous pouvez spécifier les versions de lignes à afficher. Par exemple, si la propriété **RowStateFilter** est définie à **DataViewRowState.Deleted**, le **DataView** expose la version d'**origine** de toutes les lignes **supprimées**, car ces lignes n'ont plus de version **actuelle**. Vous pouvez déterminer quelle version d'une ligne est exposée à l'aide de la propriété **RowVersion** du **DataRowView**.

Le tableau suivant présente les options de **DataViewRowState**.

DataViewRowState	Description
CurrentRows	Version **actuelle** de toutes les lignes **inchangées**, **ajoutées** et **modifiées**. Il s'agit de la valeur par défaut.
Added	Version **actuelle** de toutes les lignes **ajoutées**.
Deleted	Version d'**origine** de toutes les lignes **supprimées**.
ModifiedCurrent	Version **actuelle** de toutes les lignes **modifiées**.
ModifiedOriginal	Version d'**origine** de toutes les lignes **modifiées**.
None	Aucune ligne.
OriginalRows	Version d'**origine** de toutes les lignes **inchangées**, **modifiées** et **supprimées**.
Unchanged	Version **actuelle** de toutes les lignes **inchangées**.

Tri et filtrage de données à l'aide d'un DataView

Le **DataView** offre différentes possibilités pour le tri et le filtrage des données d'un **DataTable** :

- En utilisant la propriété **Sort**, vous pouvez spécifier un ordre de tri en fonction d'une ou de plusieurs colonnes et inclure des paramètres ASC (ordre croissant) et DESC (ordre décroissant).
- Vous pouvez utiliser la propriété **ApplyDefaultSort** pour créer automatiquement un ordre de tri, croissant, basé sur la ou les colonnes clés primaires de la table. **ApplyDefaultSort** est applicable uniquement lorsque la propriété **Sort** est une référence null ou une chaîne vide et lorsqu'une clé primaire est définie dans la table.
- En utilisant la propriété **RowFilter**, vous pouvez spécifier des sous-ensembles de lignes en fonction des valeurs de leurs colonnes.

L'exemple de code suivant crée une vue faisant apparaître tous les produits pour lesquels la quantité en stock est inférieure ou égale au niveau de passage d'une nouvelle commande, triés d'abord par ID de fournisseur, puis par nom de produit.

```
Dim prodView As DataView = New DataView(prodDS.Tables("Products"), _
                                        "UnitsInStock <= ReorderLevel", _
                                        "SupplierID, ProductName", _
                                        DataViewRowState.CurrentRows)
```

Recherche de données dans un DataView

À l'aide des méthodes **Find** et **FindRows** du **DataView**, vous pouvez rechercher des lignes en fonction des valeurs de leur clé de tri. Le respect ou le non-respect de la casse des valeurs de recherche des méthodes **Find** et **FindRows** est déterminé par la propriété **CaseSensitive** du

DataTable sous-jacent. Pour qu'un résultat soit retourné, les valeurs de recherche doivent correspondre aux valeurs des clés de tri existantes dans leur intégralité.

La méthode **Find** retourne un entier avec l'index du **DataRowView** qui correspond aux critères de recherche. Si plusieurs lignes correspondent aux critères de recherche, seul l'index du premier **DataRowView** correspondant est retourné. Si aucune correspondance n'est trouvée, **Find** retourne -1.

Pour retourner des résultats de recherche qui correspondent à plusieurs lignes, vous pouvez utiliser la méthode **FindRows**. La méthode **FindRows** fonctionne de la même manière que **Find**, à cette exception qu'elle retourne un tableau **DataRowView** faisant référence à toutes les lignes du **DataView** qui présentent une correspondance. Si aucune correspondance n'est trouvée, le tableau **DataRowView** sera vide.

Pour utiliser les méthodes **Find** ou **FindRows**, vous devez spécifier un ordre de tri soit en définissant **ApplyDefaultSort** à **true**, soit en utilisant la propriété **Sort**. Si aucun ordre de tri n'est spécifié, une exception est levée.

L'exemple de code suivant illustre la méthode **Find** appelée sur un **DataView** avec un ordre de tri défini sur une seule colonne.

```
Dim custView As DataView = New DataView(custDS.Tables("Customers"), "", _
                        "CompanyName", DataViewRowState.CurrentRows)

Dim rowIndex As Integer = custView.Find("The Cracker Box")

If rowIndex = -1 Then
  Console.WriteLine("Non trouvé.")
Else
  Console.WriteLine("{0}, {1}", _
                    custView(rowIndex)("CustomerID").ToString(), _
                    custView(rowIndex)("CompanyName").ToString())
End If
```

Si votre propriété **Sort** spécifie plusieurs colonnes, vous devez passer un tableau d'objets avec les valeurs de recherche pour chaque colonne incluse dans l'ordre spécifié par la propriété **Sort**, comme dans l'exemple de code suivant.

```
Dim custView As DataView = New DataView(custDS.Tables("Customers"), "", _
                                        "CompanyName, ContactName", _
                                        DataViewRowState.CurrentRows)

Dim foundRows() As DataRowView = custView.FindRows(New object() {"The
Cracker Box", "Liu Wong"})
```

```
If foundRows.Length = 0 Then
  Console.WriteLine("Non trouvé.")
Else
  Dim myDRV As DataRowView
  For Each myDRV In foundRows
    Console.WriteLine("{0}, {1}", myDRV("CompanyName").ToString(),
myDRV("ContactName").ToString())
  Next
End If
```

Modification de données à l'aide d'un DataView

Un **DataView** est, par défaut, une vue en lecture seule de données. Toutefois, vous pouvez utiliser le **DataView** pour ajouter, supprimer ou modifier des lignes de données dans la table sous-jacente en définissant l'une des trois propriétés de type Boolean du **DataView**. Ces propriétés sont **AllowNew**, **AllowEdit** et **AllowDelete**. Elles sont définies à **true** par défaut et vous permettent de spécifier si vous pouvez modifier les données du **DataTable** sous-jacent du **DataView**.

L'exemple de code suivant désactive la possibilité de supprimer des lignes à l'aide du **DataView** et ajoute une nouvelle ligne à la table sous-jacente à l'aide du **DataView**.

```
Dim custTable As DataTable = custDS.Tables("Customers")
Dim custView As DataView = custTable.DefaultView
custView.Sort = "CompanyName"

custView.AllowDelete = False

Dim newDRV As DataRowView = custView.AddNew()
newDRV("CustomerID") = "ABCDE"
newDRV("CompanyName") = "ABC Products"
newDRV.EndEdit()
```

Chapitre 9 : Création d'un contrôle utilisateur

Les contrôles utilisateur offrent un moyen de créer et de réutiliser des interfaces graphiques personnalisées. Un contrôle utilisateur est pour l'essentiel un composant doté d'une représentation visuelle. Il peut être constitué d'un ou de plusieurs contrôles Windows Forms, de composants ou de blocs de code, lesquels peuvent développer des fonctionnalités en validant des entrées de l'utilisateur, en modifiant des propriétés d'affichage ou en effectuant d'autres tâches imposées par le créateur du contrôle. Les contrôles utilisateur peuvent être placés dans les Windows Forms de la même façon que les autres contrôles.

Création du projet

Pour créer la bibliothèque de contrôles ctlClockLib et le contrôle ctlClock

1. Dans le menu **Fichier**, pointez sur **Nouveau**, puis sélectionnez **Projet** pour afficher la boîte de dialogue **Nouveau projet**.
2. Sélectionnez le modèle de projet **Bibliothèque de contrôles Windows** dans la liste Projets Visual Basic, et tapez **ctlClockLib** dans la zone **Nom**.
3. Recherchez l'instruction **Class**, `Public Class UserControl1`, et remplacez **UserControl1** par **ctlClock** pour changer le nom du composant.
4. Dans l'Explorateur de solutions, cliquez sur **UserControl1**, puis dans la fenêtre Propriétés, remplacez la propriété **FileName** par **ctlClock.vb**.
5. Dans le menu **Fichier**, choisissez **Enregistrer tout** pour enregistrer le projet.

Ajout de contrôles Windows et de composants à un contrôle utilisateur

Pour ajouter un contrôle Label et un contrôle Timer à votre contrôle utilisateur

1. Dans la boîte à outils, cliquez sur l'onglet **Windows Forms** puis double-cliquez sur **Label**.
2. Dans le concepteur, cliquez sur **Label1**. Dans la fenêtre Propriétés, définissez les propriétés suivantes.

Propriété	Substitution
Name	**lblDisplay**
Text	**(espace blanc)**
TextAlign	**MiddleCenter**
Font.Size	**14**

3. Dans la boîte à outils, cliquez sur l'onglet **Windows Forms** puis double-cliquez sur **Timer**.

Une minuterie étant un composant, elle n'a pas de représentation visuelle au moment de l'exécution. Elle n'est donc pas affichée avec les contrôles dans le **Concepteur de contrôles utilisateur**, mais apparaît dans la barre d'état des composants.

4. Dans la barre d'état des composants, cliquez sur **Timer1** et attribuez à la propriété **Interval** la valeur **1000** et à la propriété **Enabled** la valeur **True**.

   La propriété **Interval** détermine la fréquence des graduations du composant Timer. À chaque graduation, le composant **Timer1** exécute le code de l'événement **Timer1_Tick**. L'intervalle représente le nombre de millisecondes entre deux graduations.

5. Dans la barre d'état des composants, double-cliquez sur **Timer1** pour accéder à l'événement **Timer1_Tick** de ctlClock.
6. Modifiez le code pour qu'il ressemble à ceci :

```
Protected Sub Timer1_Tick(ByVal sender As Object, ByVal e As _
   System.EventArgs)
   ' Causes the label to display the current time
   lblDisplay.Text = Format(Now, "hh:mm:ss")
End Sub
```

   Ce code déclenche toujours l'affichage de l'heure actuelle dans **lblDisplay**. Comme l'intervalle de **Timer1** a été fixé à 1 000, cet événement est déclenché toutes les mille millisecondes, mettant ainsi à jour l'heure actuelle toutes les secondes.

7. Modifiez la méthode de telle manière qu'elle puisse être substituée.

```
Protected Overridable Sub Timer1_Tick(ByVal sender As Object, ByVal _
   e As System.EventArgs) Handles Timer1.Tick
```

8. Dans le menu **Fichier**, choisissez **Enregistrer ctlClock** pour enregistrer le projet.

Ajout de propriétés à un contrôle utilisateur

Votre contrôle d'horloge contient désormais un contrôle **Label** et un composant **Timer** dotés chacun de leurs propres propriétés inhérentes. Les propriétés de ces contrôles ne seront pas disponibles individuellement aux futurs utilisateurs de votre contrôle ; toutefois, vous pouvez créer et exposer des propriétés personnalisées en écrivant pour ce faire les blocs de code appropriés.

Pour ajouter une propriété à votre contrôle utilisateur

1. Dans l'Explorateur de solutions, cliquez avec le bouton droit sur **ctlClock.vb**, puis sur **Afficher le code** dans le menu contextuel.

2. Recherchez l'instruction `Inherits System.Windows.Forms.UserControl`. Sous cette instruction, tapez :

```
Private colFColor as Color
Private colBColor as Color
```

Ces instructions créent les variables privées que vous utiliserez pour stocker les valeurs des propriétés que vous êtes sur le point de créer.

3. Tapez le code suivant sous les déclarations de variable de l'étape 2 :

```
' Déclaration des propriétés
Property ClockBackColor() as Color
   ' Renvoie la valeur de la propriété colBColor.
   Get
      Return colBColor
   End Get

   ' Modifie la valeur de la prpriété colBColor
   Set(ByVal Value as Color)
      colBColor = Value
      lblDisplay.BackColor = ColBColor
   End Set
End Property

Property ClockForeColor() as Color
   Get
      Return colFColor
   End Get
   Set(ByVal Value as Color)
      colFColor = Value
      lblDisplay.ForeColor = ColFColor
   End Set
End Property
```

Le code précédent met deux propriétés personnalisées, ClockForeColor et ClockBackColor, à la disposition des futurs utilisateurs du contrôle en appelant l'instruction `Property`. Les instructions **Get** et **Set** assurent le stockage et l'extraction de la valeur de propriété ainsi que du code permettant d'implémenter les fonctionnalités appropriées à la propriété.

4. Dans le menu **Fichier**, choisissez **Enregistrer ctlClock.vb** pour enregistrer le projet.

Test du contrôle

Les contrôles ne sont pas des projets autonomes ; ils doivent êtes placés dans un conteneur. Pour tester votre contrôle, vous devez fournir un projet de test dans lequel il puisse s'exécuter.

Pour générer le contrôle

- Dans le menu **Générer**, cliquez sur **Générer ctlClockLib**.

Pour créer un projet de test

1. Dans le menu **Fichier**, pointez sur **Ajouter un projet** et cliquez sur **Nouveau projet** pour ouvrir la boîte de dialogue **Ajouter un nouveau projet**.
2. Cliquez sur **Application Windows**, puis tapez **Test** et cliquez sur **OK** dans la zone **Nom**.
3. Dans l'Explorateur de solutions, cliquez avec le bouton droit sur le nœud **Références** de votre projet de test. Cliquez sur **Ajouter une référence** pour afficher la boîte de dialogue **Ajouter une référence**.
4. Cliquez sur l'onglet **Projets**. Votre projet est répertorié sous **Nom du projet**.
5. Double-cliquez sur votre projet.

Une fois que vous avez ajouté votre référence, vous pouvez placer le contrôle dans votre formulaire.

Pour tester le contrôle

1. Dans la boîte à outils, cliquez sur **Windows Forms** et faites défiler la fenêtre vers le bas jusqu'à ce que s'affiche l'icône représentant le contrôle **ctlClock**.
2. Double-cliquez sur l'icône **ctlClock**.
3. Sélectionnez l'icône et amenez la souris sur votre formulaire.
4. Maintenez le bouton gauche enfoncé pendant que vous déplacez la souris sur le formulaire.
5. Dans le concepteur, cliquez sur l'une des instances de **ctlClock**.
6. Dans la fenêtre Propriétés, recherchez la propriété **ClockBackColor**, puis sélectionnez-la pour afficher la palette de couleurs.
7. Choisissez une couleur en cliquant dessus.
8. Utilisez une séquence d'événements similaire pour vérifier que la propriété **ClockForeColor** fonctionne comme prévu.

Héritage d'un contrôle utilisateur

Votre contrôle utilisateur peut être employé comme base pour la construction d'autres contrôles. Le processus consistant à dériver une classe d'une classe de base est appelé *héritage*. Dans cette section, vous créerez un contrôle utilisateur appelé ctlAlarmClock. Ce contrôle sera dérivé de son contrôle parent, ctlClock. Vous apprendrez à développer les fonctionnalités du contrôle ctlClock en substituant les méthodes parentes et en ajoutant de nouvelles méthodes et propriétés.

Création du contrôle hérité

La première étape de la création d'un contrôle hérité consiste à le dériver de son parent. Cette opération crée un contrôle qui possède toutes les propriétés, méthodes et caractéristiques graphiques du contrôle parent, mais peut également servir de base pour l'ajout de fonctionnalités nouvelles ou modifiées.

Pour créer le contrôle hérité

1. Dans l'Explorateur de solutions, cliquez sur **ctlClockLib**.
2. Dans le menu **Projet**, choisissez **Ajouter un contrôle hérité**.
3. Dans la zone **Nom**, tapez **ctlAlarmClock.vb** et cliquez sur **Ouvrir**.
4. Sous **Nom du composant**, double-cliquez sur **ctlClock**.
5. Dans l'Explorateur de solutions, parcourez les projets en cours. Notez qu'un fichier appelé **ctlAlarmClock** a été ajouté.

Ajout de propriétés d'alarme

Vous ajoutez des propriétés à un contrôle hérité comme vous le feriez pour un contrôle utilisateur. Vous allez maintenant utiliser la syntaxe de déclaration de propriété pour ajouter deux propriétés à votre contrôle : **AlarmTime**, qui stocke la valeur de date et d'heure à laquelle l'alarme se déclenche et **AlarmSet** qui indique si l'alarme est activée ou non.

Pour ajouter des propriétés à votre contrôle utilisateur

1. Dans l'Explorateur de solutions, cliquez avec le bouton droit sur **ctlAlarmClock** et sélectionnez **Afficher le code** dans le menu contextuel.
2. Recherchez l'instruction `Inherits`. Notez que votre contrôle hérite de **ctlClockLib.ctlClock**. Sous l'instruction `Inherits`, tapez le code suivant :

```
Private dteAlarmTime As Date
Private blnAlarmSet As Boolean

Public Property AlarmTime() As Date
   Get
      Return dteAlarmTime
   End Get
   Set(ByVal Value as Date)
      dteAlarmTime = Value
   End Set
End Property

Public Property AlarmSet() As Boolean
   Get
      Return blnAlarmSet
```

```
    End Get
    Set(ByVal Value as Boolean)
        blnAlarmSet = Value
    End Set
End Property
```

Ajout à l'interface graphique de votre contrôle

Votre contrôle hérité possède une interface visuelle identique à celle du contrôle dont il a hérité. Il possède les mêmes contrôles constitutifs que son contrôle parent, mais leurs propriétés ne sont pas disponibles tant qu'elles ne sont pas spécifiquement exposées. Vous pouvez faire des ajouts à une interface graphique comme vous le feriez pour n'importe quel contrôle utilisateur. Pour enrichir l'interface visuelle de votre réveil, vous allez ajouter un contrôle qui clignotera lorsque l'alarme sonnera.

Pour ajouter le contrôle Label

1. Dans l'Explorateur de solutions, cliquez avec le bouton droit sur **ctlAlarmClock** et sélectionnez **Afficher le concepteur** dans le menu contextuel.
2. Cliquez sur la partie affichage du contrôle et observez ce qui se passe dans la fenêtre Propriétés.
3. Ajoutez un contrôle **Label** à votre contrôle utilisateur.
4. À l'aide de la souris, placez le contrôle Label immédiatement sous la zone d'affichage. Dans la fenêtre Propriétés, définissez les propriétés suivantes :

Propriété	Valeur
Name	**lblAlarm**
Text	**Alarm Is Sounding!**
TextAlign	**Middle Center**
Visible	**false**

Ajout de fonctionnalités d'alarme

Vous allez ajouter du code pour comparer l'heure actuelle avec celle de l'alarme et, si les heures sont identiques, déclencher une alarme à la fois sonore et clignotante. En substituant la méthode **Timer1_Tick** de **ctlClock** et en lui ajoutant du code supplémentaire, vous développez les fonctionnalités de **ctlAlarmClock** tout en conservant les fonctionnalités inhérentes de **ctlClock**.

Pour substituer la méthode Timer1_Tick de ctlClock

1. Dans l'éditeur de code, recherchez l'instruction `Private blnAlarmSet As Boolean`. Immédiatement sous cette instruction, tapez l'instruction suivante :

```
Dim blnColorTicker as Boolean
```

2. Dans l'Éditeur de code, recherchez l'instruction `End Class`. Elle se trouve au bas de la page.
3. Juste avant l'instruction `End Class`, ajoutez le code suivant :

```
Protected Overrides Sub Timer1_Tick(ByVal sender As Object, ByVal e _
   As System.EventArgs)

   ' Appelle la méthode Timer1_Tick de ctlClock.

   MyBase.Timer1_Tick(sender, e)

   ' Vérifie si l'alarme est activée.
   If AlarmSet = False Then
      Exit Sub
   End If

   ' si date et heure = date et heure de l'alarme

   If AlarmTime.Date = Now.Date And AlarmTime.Hour = Now.Hour And _
      AlarmTime.Minute = Now.Minute Then
      Beep()
      lblAlarm.Visible = True
      If blnColorTicker = False Then
         lblAlarm.BackColor = Color.PeachPuff
         blnColorTicker = True
      Else
         lblAlarm.BackColor = Color.Plum
         blnColorTicker = False
      End If
   Else
      lblAlarm.Visible = False
   End If
End Sub
```

Votre contrôle de réveil est presque terminé. Il ne reste plus qu'à implémenter un moyen de le désactiver. Pour ce faire, vous ajoutez du code à la méthode **lblAlarm_Click**.

Pour implémenter la méthode de désactivation

1. Dans l'Explorateur de solutions, cliquez avec le bouton droit sur **ctlAlarmClock.vb**, puis cliquez sur **Concepteur de vues**.
2. Dans le concepteur, double-cliquez sur **lblAlarm**. L'**éditeur de code** s'ouvre à la ligne `Protected Sub lblAlarm_Click`.

Modifiez la méthode pour qu'elle ressemble à ceci :

```
Protected Sub lblAlarm_Click(ByVal sender As Object, ByVal e As _
 System.EventArgs) Handles lblAlarm.Click
   ' Turns off the alarm.
   AlarmSet = False
   ' Hides the flashing label.
   lblAlarm.Visible = False
End Sub
```

3. Dans le menu **Fichier**, choisissez **Enregistrer ctlAlarmClock.vb** pour enregistrer le projet.

Test du contrôle hérité

Pour générer et ajouter votre contrôle à un formulaire de test

1. Dans l'Explorateur de solutions, cliquez sur **ctlClockLib**. Dans le menu **Générer**, cliquez sur **Générer ctlClockLib**.
2. Ajoutez un nouveau projet **Application Windows** à la solution et nommez-le **Test2**.
3. Dans l'Explorateur de solutions, cliquez avec le bouton droit sur le nœud **Références** de votre projet de test. Cliquez sur **Ajouter une référence** pour afficher la fenêtre **Ajouter une référence**. Cliquez sur l'onglet **Projets**. Le projet **ctlClockLib** est répertorié sous Nom du projet. Double-cliquez sur **ctlClockLib** ; il apparaît désormais dans la fenêtre Composants sélectionnés.
4. Dans la boîte à outils, cliquez sur **Windows Forms**.
5. Faites défiler la fenêtre vers le bas jusqu'à ce que l'icône de **ctlAlarmClock** apparaisse.
6. Double-cliquez sur **ctlAlarmClock** pour ajouter une copie de **ctlAlarmClock** à votre formulaire.
7. Recherchez **DateTimePicker** et double-cliquez dessus pour ajouter un contrôle **DateTimePicker** à votre formulaire, puis ajoutez un contrôle **Label** en double-cliquant sur **Étiquette**.
8. À l'aide de la souris, placez les contrôles dans le formulaire à un endroit adéquat.
9. Définissez comme suit les propriétés de ces contrôles :

Contrôle	Propriété	Value
Label1	**Text**	(laisser vide)
	Name	**lblTest**
DateTimePicker1	**Name**	**dtpTest**
	Format	**Time**

10. Dans le concepteur, double-cliquez sur **dtpTest**.

    L'**Éditeur de code** s'ouvre sur `Protected Sub dtpTest_ValueChanged`.

11. Modifiez le code pour qu'il ressemble à ceci :

```
Protected Sub dtpTest_ValueChanged(ByVal sender As Object, ByVal e As
_
   System.EventArgs) Handles dtpTest.ValueChanged
   ctlAlarmClock1.AlarmTime = dtpTest.Value
   ctlAlarmClock1.AlarmSet = True
   lblTest.Text = "Alarm Time is " & Format(ctlAlarmClock1.AlarmTime,
_
      "hh:mm")
End Sub
```

12. Dans l'Explorateur de solutions, cliquez avec le bouton droit sur **Test2** puis sélectionnez **Définir comme projet de démarrage** dans le menu contextuel.
13. Dans le menu **Déboguer**, cliquez sur **Démarrer**.

    Le programme de test démarre. Notez que l'heure actuelle est mise à jour dans le contrôle **ctlAlarmClock** et que l'heure de démarrage est affichée dans le contrôle **DateTimePicker**.

14. Cliquez sur le contrôle **DateTimePicker** dans lequel sont affichées les minutes de l'heure.
15. À l'aide du clavier, définissez pour les minutes une valeur supérieure d'une minute à l'heure actuelle affichée dans **ctlAlarmClock**.

    L'heure de définition de l'alarme apparaît dans **lblTest**.

16. Attendez que l'heure affichée atteigne l'heure de définition de l'alarme.

    Lorsque l'heure affichée atteint celle à laquelle l'alarme est définie, le signal sonore se fait entendre et **lblAlarm** se met à clignoter. Désactivez l'alarme en cliquant sur **lblAlarm**. Vous pouvez maintenant réinitialiser l'alarme.

Chapitre 10 : Création d'états Crystal Reports

Crystal Reports est l'outil de création d'états fourni avec Visual Studio .NET. Nous allons voir comment créer un état alimenté par des données extraites d'une application .NET et comment afficher l'état généré dans une fenêtre Windows.

Création du dataset typé

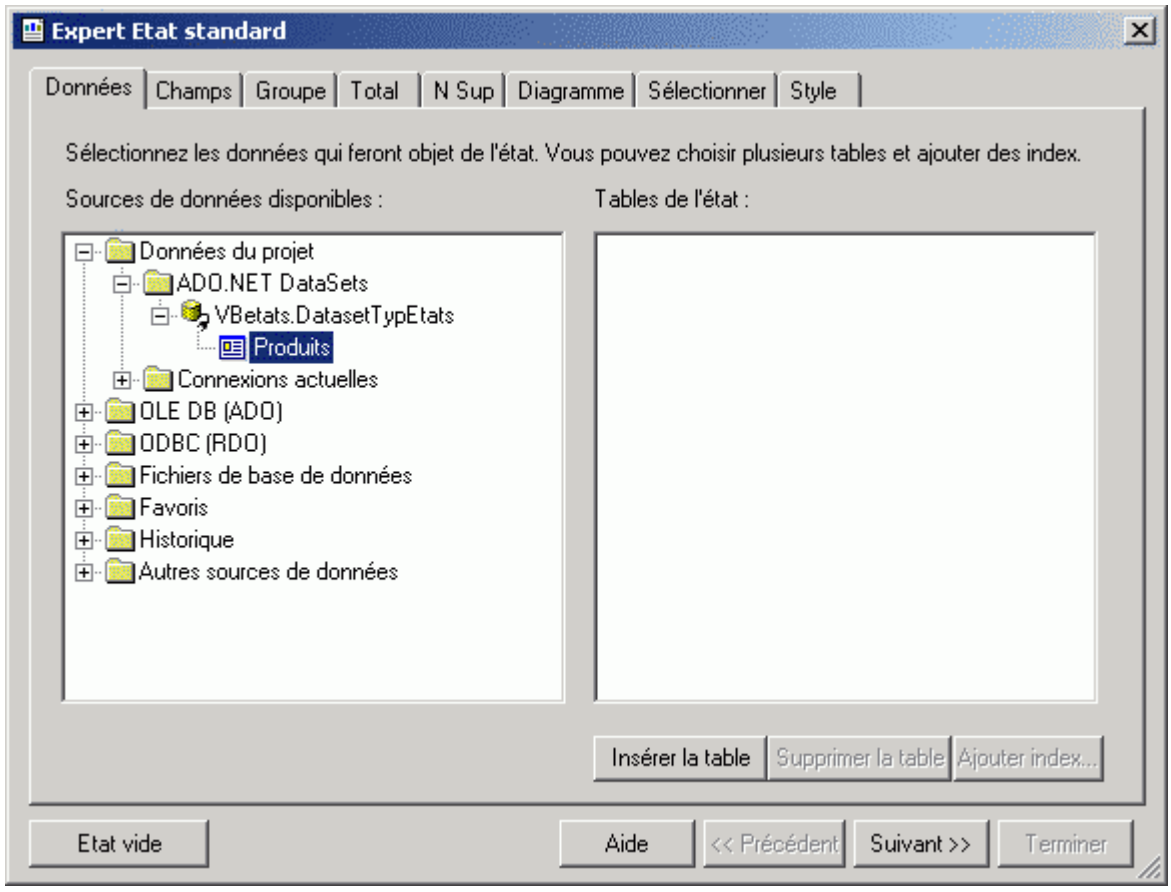
Nous devons commencer par créer un dataset typé. Pour que notre état puisse consommer les données de notre dataset celui-ci doit être typé. En effet, lors de la création de l'état il faut bien que Crystal Reports puisse déterminer le nom et le type des champs pour les données qui lui serviront à générer le rapport.

Création de l'état

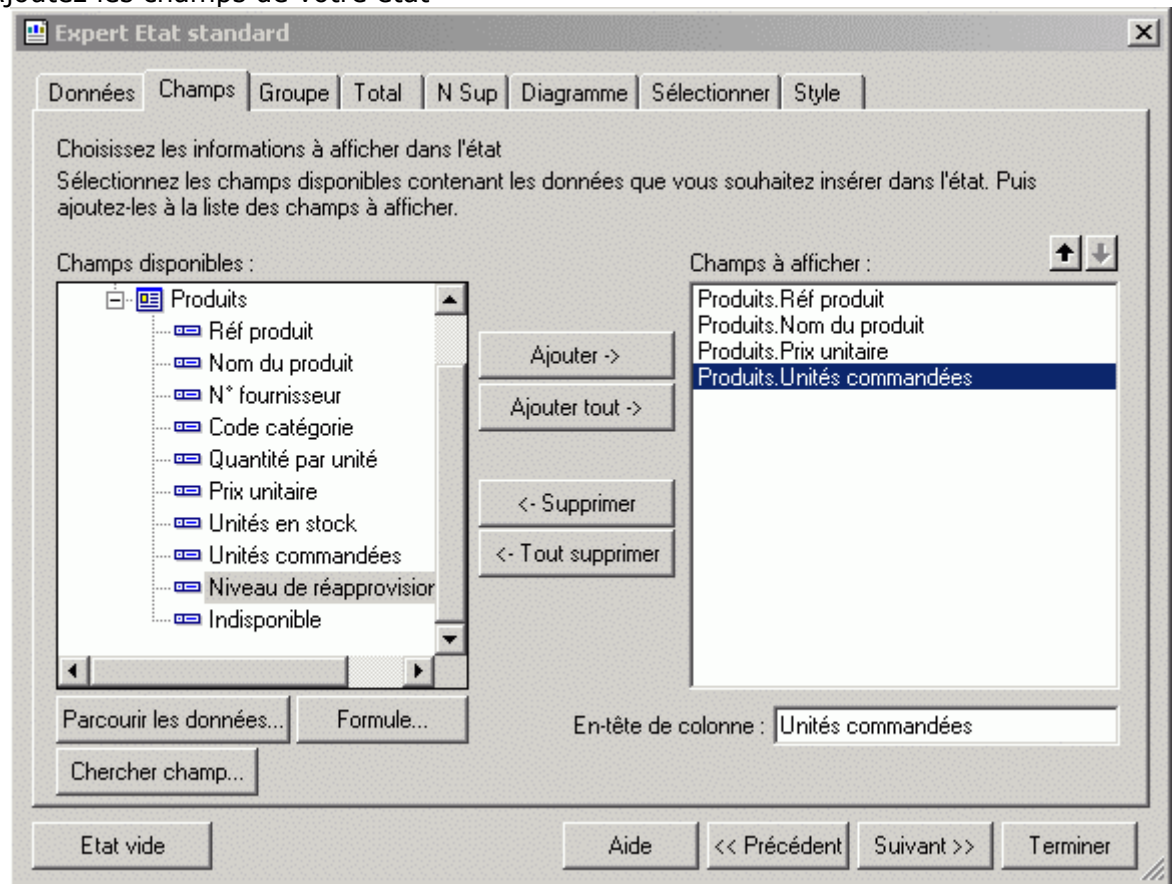
Tout d'abord il faut ajouter un état Crystal Reports au projet.

1. Dans l'Explorateur de solutions Visual Studio .NET, cliquez avec le bouton droit de la souris sur votre projet pour afficher le menu contextuel.
2. Pointez sur **Ajouter** et cliquez sur **Ajouter un nouvel élément**.
3. Dans la boîte de dialogue Ajouter un nouvel élément, sélectionnez **Etat Crystal Reports** dans la zone Modèles. Cliquez sur **Ouvrir**.
4. Dans la Collection d'états Crystal Reports, sélectionnez une des options suivantes :
    - **Utilisation de l'Expert Etat** – vous guide tout au long du processus de création d'état et ajoute vos choix au Crystal Report Designer.
    - **En tant qu'état vide** – ouvre le Crystal Report Designer.
    - **Dans un état existant** – crée un nouvel état avec la même conception qu'un autre état que vous spécifiez.
5. Cliquez sur **OK**.

Nous allons ensuite utiliser l'expert d'état pour créer rapidement notre état.
Dans l'onglet "Données" développer l'arbre ADO.NET, puis le dataset



Cliquez sur "Insérer la table"
Ajoutez les champs de votre état



Passer directement à l'onglet "Diagramme". Les onglets "Groupe" "Total" et "Nsup" permettent de classer les données et d'afficher des totaux par colonne et par groupe. Nous allons réaliser un diagramme sectoriel sur la quantité de produits commandés.

- Choisir un diagramme de type "sectoriel", faire "suivant".
- Sélectionner "sur changement" puis ajouter le champ "Réf produit"
- Sélectionner le champ "Unités commandées" dans "Afficher des valeurs".
- Faire "Terminer".

Voilà, notre état Crystal Reports est enfin créé. Voyons maintenant comment générer les rapports.

Visualisation d'un état dans une application windowsform

On doit commencer par créer une instance de notre état crystal report.

```
Dim rpt As New CrystalReport1
```

Maintenant voyons comment passer les données du dataset à notre crystal report et comment le visualiser à l'aide du viewer.

```
rpt.SetDataSource(dsNorthwind1)
CrystalReportViewer1.ReportSource = rpt
CRviewer.Refresh()
CrystalReportViewer1.Clear ()
```

Chapitre 11 : Gestion d'une exception d'accès concurrentiel

Vous allez créer une application Windows qui déclenche une erreur d'accès concurrentiel et illustre une stratégie de gestion de cette erreur. Vous allez simuler le travail de deux utilisateurs sur les mêmes données au même moment. Le Windows Form que vous allez créer vous permet de travailler en tant que l'un ou l'autre des deux utilisateurs depuis le même formulaire.

1. L'utilisateur 1 et l'utilisateur 2 remplissent leurs groupes de données avec les mêmes données.
2. L'utilisateur 2 modifie un enregistrement, met à jour le groupe de données et écrit les modifications dans la source de données.
3. L'utilisateur 1 modifie le même enregistrement, modifie le groupe de données et tente d'écrire les modifications dans la source de données. Ceci déclenche une erreur d'accès concurrentiel.

Vous allez détecter cette erreur, puis afficher les différentes versions de l'enregistrement, afin de permettre à l'utilisateur de déterminer comment doivent être traitées les modifications en attente de l'utilisateur 1.

Création d'un projet et d'une connexion de données

Pour créer un projet

1. Dans le menu **Fichier**, pointez sur **Nouveau**, puis cliquez sur **Projet** pour afficher la boîte de dialogue **Nouveau projet**.
2. Nommez le projet **acces_concurrentiel**, puis cliquez sur **OK**.

Pour créer une connexion de données

1. Dans l'Explorateur de serveurs, créez une connexion à l'exemple de base de données Pubs.
2. Dans l'Explorateur de serveurs, développez la connexion que vous avez créée lors de l'étape précédente.
3. Développez la zone **Tables**.
4. Faites glisser la table **authors** vers votre formulaire.

   Un objet **Connection** et un objet **DataAdapter** apparaissent dans la barre des composants située sous le formulaire.

Création des groupes de données

Vous allez créer deux groupes de données appelés **DsAuthors1** et **DsAuthors2**. Ces groupes de données représenteront les données sur lesquelles les deux utilisateurs travaillent

simultanément. Vous ajouterez ensuite au formulaire deux contrôles DataGrid que vous allez lier aux groupes de données. Enfin, deux contrôles Button seront ajoutés au formulaire : un bouton **Mettre à jour** et un bouton **Rétablir**.

Pour créer deux groupes de données

1. Sélectionnez l'objet **DataAdapter**.
2. Dans le menu **Données**, sélectionnez **Générer un DataSet**.
3. Sélectionnez **Nouveau** et attribuez au groupe de données le nom **DsAuthors**.
4. À partir de l'onglet **Données** de la **Boîte à outils**, faites glisser un **DataSet** jusqu'au formulaire.
5. Vérifiez que **DataSet typé** est sélectionné et que **acces_concurrentiel.DsAuthors** apparaît dans la zone **Nom**.

   Une instance appelée **DsAuthors2** apparaît dans la barre d'état des composants du concepteur.

Liaison des données et ajout des boutons

Pour ajouter deux DataGrids au formulaire

1. À partir de l'onglet **Windows Forms** de la Boîte à outils, faites glisser un objet **DataGrid** jusqu'au côté gauche de votre formulaire.
2. À partir de l'onglet **Windows Forms** de la Boîte à outils, faites glisser un objet **DataGrid** jusqu'au côté droit de votre formulaire.

Pour lier les groupes de données aux contrôles DataGrid

1. Sélectionnez **DataGrid1** et définissez les propriétés suivantes dans la fenêtre Propriétés :

Propriété	Valeur
DataSource	**DsAuthors1**
DataMember	**authors**
CaptionText	**DsAuthors1**

2. Sélectionnez **DataGrid2** et définissez les propriétés suivantes dans la fenêtre Propriétés :

Propriété	Valeur
DataSource	**DsAuthors2**
DataMember	**Authors**
CaptionText	**DsAuthors2**

Pour ajouter les boutons Mettre à jour et Rétablir au formulaire

1. À partir de l'onglet **Windows Forms** de la Boîte à outils, faites glisser un contrôle **Button** jusqu'au formulaire et placez-le au-dessus de **DataGrid1**.
2. Sélectionnez le bouton. Dans la fenêtre Propriétés, nommez le bouton **btnUpdate** et définissez sa propriété **Text** sur **Mettre à jour**.
3. À partir de l'onglet **Windows Forms** de la Boîte à outils, faites glisser un second objet **Button** jusqu'au formulaire et placez-le au-dessus de **DataGrid2**.
4. Sélectionnez le bouton. Dans la fenêtre Propriétés, nommez le bouton **btnReset** et définissez sa propriété **Text** sur **Rétablir**.

Remplissage des groupes de données

Cette étape consiste à remplir les deux groupes de données avec les mêmes données.

Pour ajouter le code permettant de remplir les groupes de données à partir de la base de données

Insérez le code suivant dans l'Éditeur de code :

```
Private Sub filltheDataSets()
   SqlDataAdapter1.Fill(DsAuthors1)
   SqlDataAdapter1.Fill(DsAuthors2)
End Sub
```

Simulation des modifications de l'utilisateur 2

Pour ajouter le code permettant de simuler les modifications de l'utilisateur 2

Insérez le code suivant dans l'Éditeur de code :

```
Private Sub user2changes()
   DsAuthors2.authors(0).au_fname = "User 2"
   SqlDataAdapter1.Update(DsAuthors2.GetChanges())
   SqlDataAdapter1.Fill(DsAuthors2)
End Sub
```

Création du gestionnaire d'événements Form_Load

Pour ajouter dans l'événement Form_Load le code permettant d'initialiser la procédure

Double-cliquez sur une zone vide du formulaire pour créer automatiquement le gestionnaire d'événements **Form_Load**.

Ajoutez du code pour que le gestionnaire d'événements ressemble à ceci :

```
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
   filltheDataSets()
   user2changes()
End Sub
```

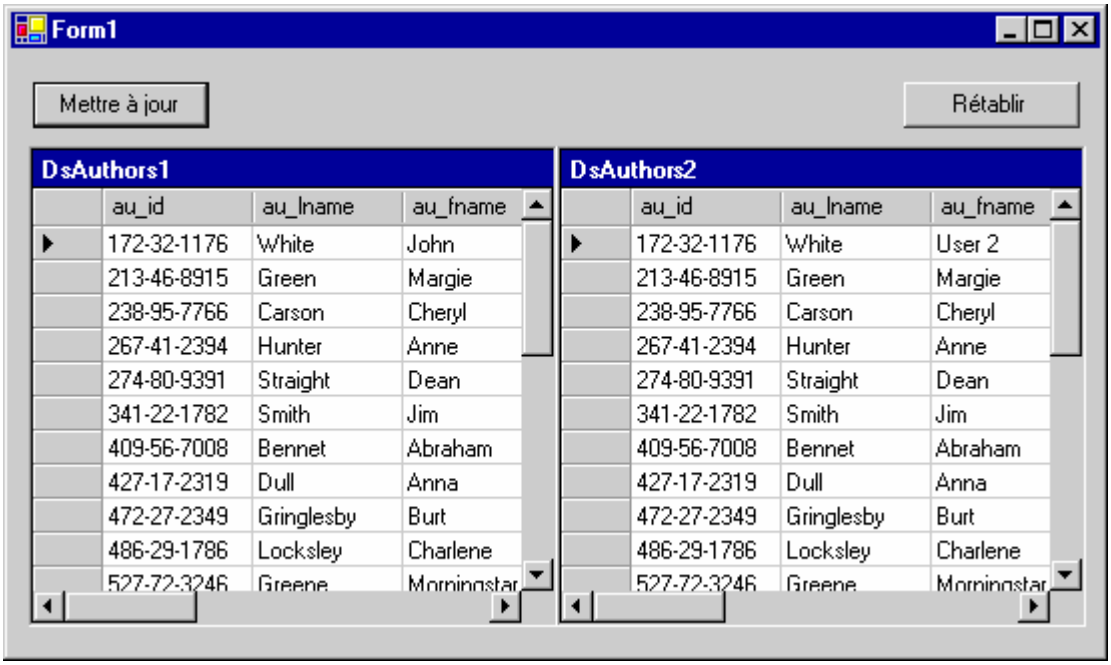
Enregistrez votre projet.

Exécution de l'application

1. Appuyez sur **F5** pour exécuter l'application.

   Le formulaire apparaît avec deux datagrids remplis avec les données de la table authors de la base de données Pubs. Le champ `au_fname` du premier enregistrement de DsAuthors1 doit être **John**. Le champ `au_fname` du premier enregistrement de DsAuthors2 doit être **User 2**.

   Votre formulaire devrait ressembler à ceci :



2. Dans le menu **Déboguer**, cliquez sur **Arrêter le débogage**.

Mise à jour du groupe de données et écriture des modifications dans la base de données

Vous allez ensuite écrire le code qui va tenter de mettre à jour la base de données avec les modifications du groupe de données **DsAuthors1**.

Pour mettre à jour la base de données

1. Appelez la méthode Update.
2. Créez un gestionnaire d'exceptions qui affiche un message.

Votre code doit se présenter comme suit :

```
Private Sub updateDatabase()
    Try
        SqlDataAdapter1.Update(DsAuthors1.GetChanges)
        DsAuthors1.AcceptChanges()
```

```
            MessageBox.Show("Mise à jour effectuée avec succès!")
        Catch ex As Exception
            MessageBox.Show(ex.Message, ex.GetType.ToString)
        End Try
    End Sub
```

Ensuite, vous allez ajouter le code qui modifie la colonne au_fname dans le groupe de données DsAuthors1. Le code appelle ensuite la procédure **updateDatabase** pour tenter d'écrire cette modification dans la base de données. Comme cette valeur a déjà été modifiée par l'utilisateur 2, une erreur d'accès concurrentiel est déclenchée.

Pour mettre à jour le groupe de données DsAuthors1

1. Double-cliquez sur le bouton **Mettre à jour**.
2. Créez le gestionnaire d'événements **btnUpdate_Click** :

```
Private Sub btnUpdate_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles btnUpdate.Click
   DsAuthors1.authors(0).au_fname = "User 1"
   updateDatabase()
End Sub
```

3. Enregistrez votre projet.
4. Appuyez sur **F5** pour exécuter l'application.
5. Cliquez sur le bouton **Mettre à jour** pour définir le champ **au_fname** du premier enregistrement sur **User 1**.

   L'erreur d'accès concurrentiel est déclenchée.

Gestion des erreurs d'accès concurrentiel

La façon dont vous gérez l'erreur dépend des règles spécifiques à votre entreprise, qui gèrent votre application. La stratégie de gestion d'erreur ci-dessous sera utilisée comme illustration. L'application présentera à l'utilisateur trois versions de l'enregistrement :

- L'enregistrement en cours dans la base de données
- L'enregistrement d'origine dans le groupe de données
- Les modifications proposées dans le groupe de données

L'utilisateur aura alors la possibilité d'écrire les modifications proposées dans la base de données ou d'annuler la mise à jour et d'actualiser le groupe de données.

Création d'un gestionnaire d'erreurs personnalisé

Lorsque vous effectuez une mise à jour, il est souvent préférable d'utiliser un gestionnaire d'exceptions structuré qui vous permet de détecter les erreurs. Dans le code que vous avez utilisé plus haut, vous avez inclus un bloc **try...catch** qui détecte toutes les erreurs — c'est-à-dire une structure qui contient une instruction **catch** générique pour toutes les erreurs.

Vous pouvez également détecter des erreurs spécifiques de façon à réagir de manière appropriée. À titre d'illustration, cette procédure va ajouter un gestionnaire d'exceptions pour une erreur spécifique, à savoir une erreur d'accès concurrentiel que vous pouvez examiner en utilisant l'objet DbConcurrencyException. Vous allez gérer cette erreur par l'affichage d'informations à l'attention de l'utilisateur.

Pour ajouter une gestion spécifique de l'erreur DBConcurrencyException

1. Si l'application est toujours en cours d'exécution, quittez le mode exécution pour revenir à l'Éditeur de code.
2. Ajoutez une deuxième instruction **catch** au-dessus de l'instruction existante dans la méthode **updateDatabase**.
3. Passez l'objet **DBConcurrencyException** à la procédure `createMessage` que vous allez créer dans la prochaine section.

```
Private Sub updateDatabase()
    Try
        SqlDataAdapter1.Update(DsAuthors1.GetChanges())
        DsAuthors1.AcceptChanges()
        MessageBox.Show("Mise à jour effectuée avec succès!")
    Catch dbcx As DBConcurrencyException
        createMessage(dbcx)
    Catch ex As Exception
        MessageBox.Show(ex.Message, ex.GetType().ToString())
    End Try
End Sub
```

Affichage des choix pour l'utilisateur

Le code que vous venez de taper appelle la procédure `createMessage` qui va afficher les informations sur l'erreur pour l'utilisateur. Dans cette procédure, vous utiliserez une boîte de message pour afficher les différentes versions de l'enregistrement pour l'utilisateur et lui permettre de choisir entre le remplacement de l'enregistrement avec les modifications et l'annulation des modifications.

Pour créer la procédure createMessage

Créez le gestionnaire d'erreurs en ajoutant le code ci-dessous dans l'Éditeur de code :

```
Private Sub createMessage(ByVal dbcx As DBConcurrencyException)
   Dim strInDs As String = "Enregistrement d'origine dans DsAuthors1:" _
      & ControlChars.CrLf
   Dim strInDB As String = "Enregistrement Courant dans la base:" _
      & ControlChars.CrLf
   Dim strProposed As String = "Modification proposée:" &
ControlChars.CrLf
   Dim strPromptText As String = "Voulez vous remplacez l'enregistrement
courant?" & ControlChars.CrLf
   Dim strMessage As String
   Dim reponse As System.Windows.Forms.DialogResult
```

```
   Dim rowInDB As DataRow = _
      DsAuthors2.authors.FindByau_id(CType(dbcx.Row("au_id"), String))

   Dim i As Integer
   For i = 0 To dbcx.Row.ItemArray.Length - 1
      strInDs &= dbcx.Row(i, DataRowVersion.Original) & _
         ControlChars.Tab
      strInDB &= rowInDB(i, DataRowVersion.Current) & ControlChars.Tab
      strProposed &= dbcx.Row(i, DataRowVersion.Current) & _
         ControlChars.Tab
   Next

   strMessage = strInDs & ControlChars.CrLf
   strMessage &= strInDB & ControlChars.CrLf
   strMessage &= strProposed & ControlChars.CrLf
   strMessage &= strPromptText

   reponse = MessageBox.Show(strMessage, dbcx.Message, _
      MessageBoxButtons.YesNo)
   processResponse(reponse)
End Sub
```

Traitement de la réponse de l'utilisateur

Vous devrez également ajouter du code pour le traitement de la réponse de l'utilisateur au message. Il pourra choisir de remplacer ou non l'enregistrement actuel de la base de données par la modification proposée. Si l'utilisateur choisit d'effectuer le remplacement, la méthode Merge de **DsAuthors1** est appelée avec l'argument *preserveChanges* défini sur **true**. Les versions d'origine des lignes de données de **DsAuthors2** sont alors fusionnées avec les versions actuelles des lignes de données de **DsAuthors1**. La mise à jour fonctionnera alors correctement car la version d'origine de l'enregistrement correspond désormais à la base de données.

Pour traiter la réponse de l'utilisateur au message

Ajoutez le code suivant dans l'Éditeur de code :

```
Private Sub processResponse(ByVal response As _
System.Windows.Forms.DialogResult)
   Select Case reponse
      Case System.Windows.Forms.DialogResult.Yes
         DsAuthors1.Merge(DsAuthors2, True)
         SqlDataAdapter1.Update(DsAuthors1)
         DsAuthors1.AcceptChanges()
      Case System.Windows.Forms.DialogResult.No
         DsAuthors1.Merge(DsAuthors2)
   End Select
End Sub
```

Chapitre 12 : Gestion des transactions

A) Gestion des transactions au niveau SQL server

1) Les transactions locales

BEGIN TRANSACTION

Indique le début d'une transaction locale explicite. L'instruction BEGIN TRANSACTION incrémente de 1 la variable @@TRANCOUNT.

Syntaxe

BEGIN TRAN [SACTION] [*transaction_name* | *@tran_name_variable*
[WITH MARK ['*description*']]]

Arguments

transaction_name

Nom attribué à la transaction. *transaction_name* doit respecter les règles applicables aux identificateurs, mais seuls les 32 premiers caractères de ce nom sont utilisés. Utilisez les noms de transaction seulement sur la paire la plus extérieure des instructions BEGIN...COMMIT ou BEGIN...ROLLBACK imbriquées.

@tran_name_variable

Nom d'une variable définie par l'utilisateur et contenant un nom de transaction valide. La variable doit être déclarée avec un type de données **char**, **varchar**, **nchar** ou **nvarchar**.

WITH MARK ['*description*']

Indique que la transaction est marquée dans le journal. *description* est une chaîne qui décrit la marque.

Si WITH MARK est utilisé, un nom de transaction doit être spécifié. WITH MARK permet de restaurer un journal de transactions par rapport à une marque nommée.

Notes

BEGIN TRANSACTION représente le point auquel les données référencées par une connexion sont logiquement et physiquement cohérentes. Si des erreurs sont détectées, toutes les modifications apportées aux données après l'exécution de l'instruction BEGIN TRANSACTION peuvent être annulées afin que les données reviennent à cet état connu de cohérence. Une transaction se poursuit jusqu'à ce qu'elle se termine sans erreur et que l'instruction COMMIT TRANSACTION soit émise pour que les modifications soient intégrées de façon permanente à la base de données ou bien jusqu'à ce que des erreurs soient

rencontrées, auquel cas les modifications sont effacées à l'aide de l'instruction ROLLBACK TRANSACTION.

Transactions marquées

L'option WITH MARK permet de placer le nom de la transaction dans le journal des transactions. Lors de la restauration d'une base de données dans un état antérieur, la transaction marquée peut être utilisée à la place d'une date et d'une heure.

L'instruction BEGIN TRAN new_name WITH MARK peut être imbriquée dans une transaction existante non marquée. *new_name* devient alors le nom de marque de la transaction, malgré le nom éventuellement déjà affecté à la transaction. Dans l'exemple suivant, M2 est le nom de la marque.

```
BEGIN TRAN T1
UPDATE table1 ...
BEGIN TRAN M2 WITH MARK
UPDATE table2 ...
SELECT * from table1
COMMIT TRAN M2
UPDATE table3 ...
COMMIT TRAN T1
```

Si vous essayez de marquer une transaction déjà marquée, vous obtenez un message d'avertissement (non d'erreur) :

```
BEGIN TRAN T1 WITH MARK
UPDATE table1 ...
BEGIN TRAN M2 WITH MARK


Server: Msg 3920, Level 16, State 1, Line 3
WITH MARK option only applies to the first BEGIN TRAN WITH
MARK.
The option is ignored.
```

Autorisations

L'autorisation d'exécuter l'instruction BEGIN TRANSACTION est attribuée par défaut à tout utilisateur reconnu.

COMMIT TRANSACTION

Marque la fin d'une transaction réussie. Si @@TRANCOUNT vaut 1, COMMIT TRANSACTION rend permanentes les modifications de données effectuées dans la base de

données depuis le début de la transaction, libère les ressources utilisées par la connexion et décrémente @@TRANCOUNT à 0. Si @@TRANCOUNT est supérieur à 1, COMMIT TRANSACTION décrémente @@TRANCOUNT seulement de 1.

Syntaxe

COMMIT [TRAN [SACTION] [*transaction_name* | *@tran_name_variable*]]

Arguments

transaction_name

Argument ignoré par Microsoft® SQL Server™. Il est utilisé pour faciliter la lecture par des programmeurs en indiquant à quelle instruction BEGIN TRANSACTION imbriquée l'instruction COMMIT TRANSACTION est associée.

@tran_name_variable

Nom d'une variable définie par l'utilisateur et contenant un nom de transaction valide. La variable doit être déclarée avec un type de données **char**, **varchar**, **nchar** ou **nvarchar**.

Exemple

A. Validation d'une transaction.

L'exemple suivant augmente l'acompte versé à un auteur lorsque les ventes d'un livre pour l'année en cours sont supérieures à 8000 FF.

```
BEGIN TRANSACTION
USE pubs
GO
UPDATE titles
SET advance = advance * 1.25
WHERE ytd_sales > 8000
GO
COMMIT
GO
```

B. Validation d'une transaction imbriquée.

L'exemple suivant crée une table, génère trois niveaux de transactions imbriquées, puis valide la transaction imbriquée. Bien que chaque instruction COMMIT TRANSACTION ait un paramètre *transaction_name*, il n'existe aucune relation entre les instructions COMMIT TRANSACTION et BEGIN TRANSACTION. Les paramètres *transaction_name* aident simplement le programmeur à s'assurer qu'un nombre suffisant de validations a été codé pour réduire @@TRANCOUNT à 0, ce qui aura pour effet de valider la transaction externe.

```
CREATE TABLE TestTran (Cola INT PRIMARY KEY, Colb CHAR(3))
GO
BEGIN TRANSACTION OuterTran -- @@TRANCOUNT set to 1.
GO
INSERT INTO TestTran VALUES (1, 'aaa')
GO
BEGIN TRANSACTION Inner1 -- @@TRANCOUNT set to 2.
GO
INSERT INTO TestTran VALUES (2, 'bbb')
GO
BEGIN TRANSACTION Inner2 -- @@TRANCOUNT set to 3.
GO
INSERT INTO TestTran VALUES (3, 'ccc')
GO
COMMIT TRANSACTION Inner2 -- Decrements @@TRANCOUNT to 2.
-- Nothing committed.
GO
COMMIT TRANSACTION Inner1 -- Decrements @@TRANCOUNT to 1.
-- Nothing committed.
GO
COMMIT TRANSACTION OuterTran -- Decrements @@TRANCOUNT to 0.
-- Commits outer transaction OuterTran.
GO
```

ROLLBACK TRANSACTION

Annule une transaction implicite ou explicite jusqu'au début de la transaction ou jusqu'au dernier point d'enregistrement à l'intérieur de la transaction.

2) Les transactions distribuées

Les transactions distribuées sont réparties sur plusieurs serveurs nommés gestionnaires de ressources. La gestion de la transaction est coordonnée entre les gestionnaires de ressources par un composant du serveur nommé gestionnaire de transactions Microsoft Distributed Transaction Coordinator (MS DTC).

Une transaction exécutée sur un seul serveur SQL Server mais utilisant plusieurs bases de données est en réalité une transaction distribuée. SQL Server, toutefois, gère la transaction distribuée de manière interne ; elle apparaît comme une transaction locale pour l'utilisateur.

Dans une application, une transaction distribuée est gérée de manière comparable à une transaction locale. À la fin de la transaction, l'application requiert que la transaction soit validée ou annulée. La validation d'une transaction distribuée doit être gérée de façon particulière par le gestionnaire de transaction pour minimiser les risques qu'une défaillance du réseau entraîne la validation de la transaction par certains gestionnaires de ressources, alors qu'elle sera annulée par d'autres. Pour cela, le processus de validation est géré suivant deux phases (une phase de préparation et une phase de validation), et est connu sous le nom de validation à deux phases (2PC).

Phase de préparation

Lorsque le gestionnaire de transaction reçoit une requête de validation, il envoie une commande de préparation à tous les gestionnaires de ressources impliqués dans la transaction. Chaque gestionnaire de ressources effectue alors toutes les opérations nécessaires à l'enregistrement de la transaction, et tous les tampons contenant les images du journal de la transaction sont vidés sur le disque. Lorsque chaque gestionnaire de ressources a terminé la phase de préparation, il renvoie un message de succès ou d'échec au gestionnaire de transactions.

Phase de validation

Si le gestionnaire de transactions reçoit des messages de préparation réussie de tous les gestionnaires de ressources, il envoie une commande de validation à chacun d'entre eux. Les gestionnaires de ressources peuvent alors effectuer la validation. Si tous les gestionnaires de ressources indiquent le succès de la validation, le gestionnaire de transactions envoie alors une notification de succès à l'application. Si l'un des gestionnaires de ressources indique un échec de la préparation, le gestionnaire de transactions envoie une commande d'annulation à chaque gestionnaire de ressources et notifie l'échec de la validation à l'application.

BEGIN DISTRIBUTED TRANSACTION

Indique le début d'une transaction Transact-SQL distribuée gérée par MS DTC (Microsoft Distributed Transaction Coordinator).

Syntaxe

BEGIN DISTRIBUTED TRAN [SACTION]
  [*transaction_name* | *@tran_name_variable*]

Le serveur exécutant l'instruction BEGIN DISTRIBUTED TRANSACTION est le créateur de la transaction et c'est aussi lui qui contrôle l'exécution jusqu'à son terme. Si une instruction COMMIT TRANSACTION ou ROLLBACK TRANSACTION est ensuite émise pour la connexion, le serveur de contrôle demande à MS DTC de gérer l'exécution de la transaction distribuée sur tous les serveurs concernés.

Deux méthodes permettent d'inscrire des serveurs SQL distants dans une transaction distribuée :

- Une connexion déjà inscrite dans la transaction distribuée effectue un appel de procédure distante faisant référence à un serveur distant.
- Une connexion déjà inscrite dans la transaction distribuée exécute une requête distribuée faisant référence à un serveur distant.

Par exemple, si l'instruction BEGIN DISTRIBUTED TRANSACTION est émise sur un **ServerA**, la connexion appelle une procédure stockée sur un **ServerB** et une autre sur un **ServerC**, et la procédure stockée sur le **ServerC** exécute une requête distribuée sur un **ServerD**, à la suite de quoi les quatre serveurs SQL se retrouvent inscrits dans la transaction distribuée. **ServerA** est le serveur qui crée la transaction et contrôle son exécution.

Les connexions intervenant dans les transactions Transact-SQL distribuées n'obtiennent pas d'objet de transaction qu'elles peuvent transmettre à une autre connexion pour que celle-ci soit explicitement inscrite dans la transaction distribuée. La seule façon pour un serveur distant de s'inscrire dans une transaction est d'être la cible d'un appel de procédure stockée distante ou d'une requête distribuée.

Exemple

Dans l'exemple suivant, le nom de l'auteur est mis à jour dans les bases de données locales et distantes. Ces bases vont soit valider soit annuler la transaction.

> **Remarque** Cet exemple provoque un message d'erreur, sauf si MS DTC est actuellement installé sur l'ordinateur exécutant SQL Server. Pour plus d'informations sur l'installation de MS DTC, voir la documentation Microsoft Distributed Transaction Coordinator.

```
USE pubs
GO
BEGIN DISTRIBUTED TRANSACTION
UPDATE authors
   SET au_lname = 'McDonald' WHERE au_id = '409-56-7008'
EXECUTE remote.pubs.dbo.changeauth_lname '409-56-
7008','McDonald'
COMMIT TRAN
GO
```

Le service MS DTC

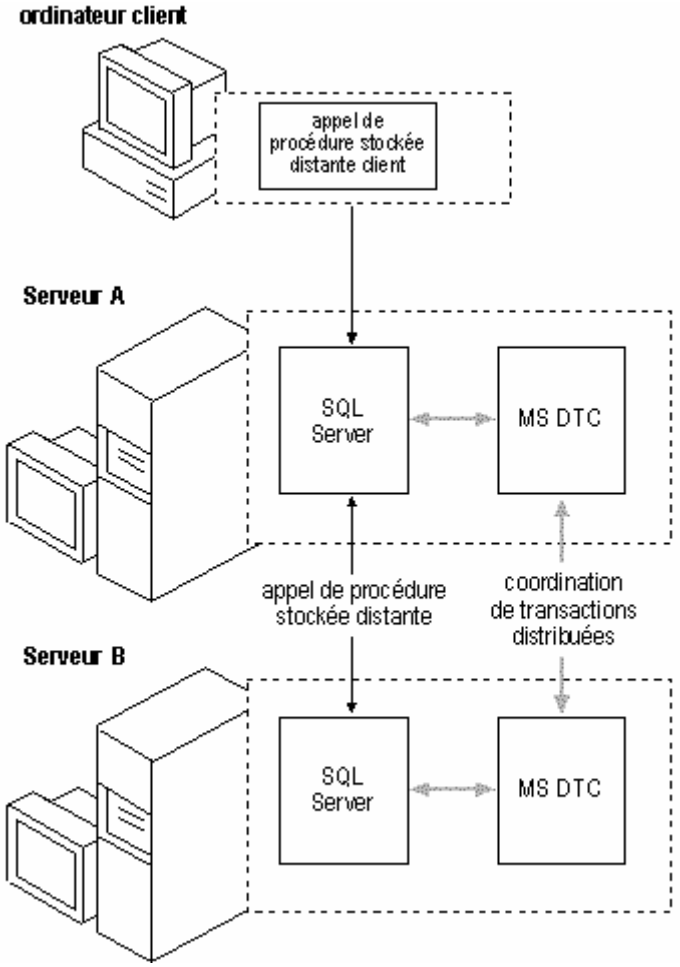
Microsoft Distributed Transaction Coordinator (MS DTC) est un gestionnaire de transactions qui permet aux applications clientes d'inclure plusieurs sources de données différentes dans une seule transaction. MS DTC coordonne la validation de la transaction distribuée au travers de tous les serveurs inscrits dans la transaction.

Une installation Microsoft® SQL Server™ peut participer à une transaction distribuée en :

- Appelant des procédures stockées sur des serveurs distants exécutant SQL Server.
- Transformant automatiquement ou explicitement la transaction locale en une transaction distribuée et en inscrivant les serveurs distants dans la transaction.
- Effectuant des mises à jour distribuées qui mettent à jour les données sur plusieurs sources de données OLE DB.

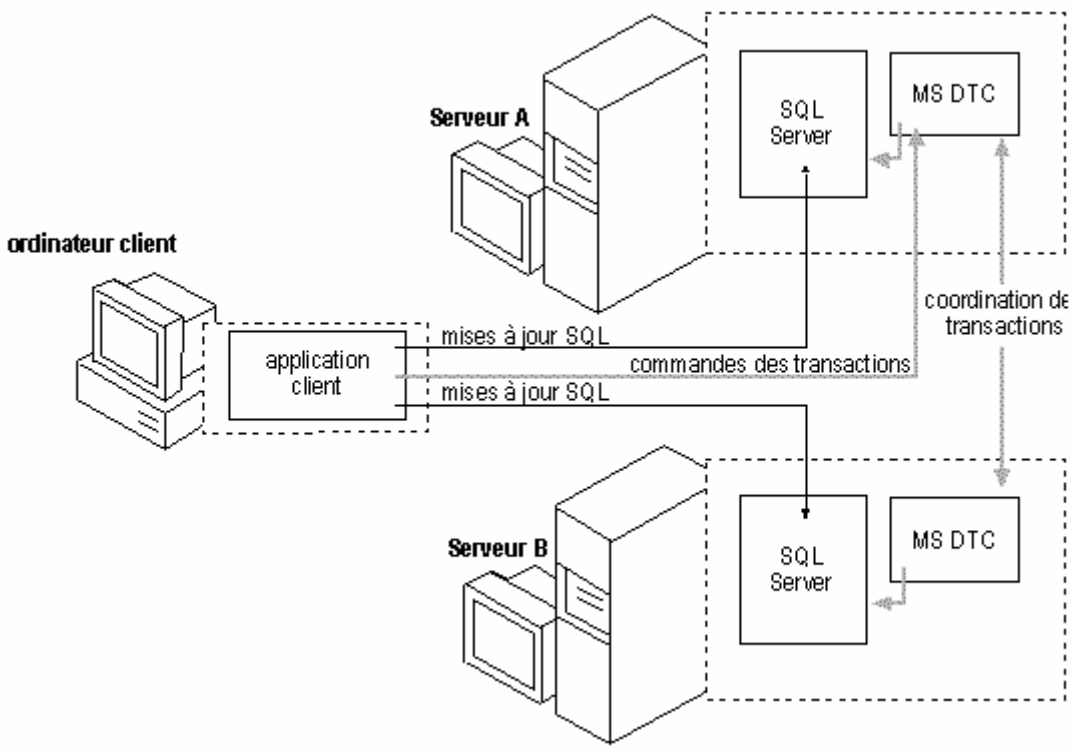
  Si ces sources de données OLE DB prennent en charge l'interface de transaction distribuée OLE DB, SQL Server peut également les inscrire dans une transaction distribuée.

MS DTC est le service qui coordonne l'achèvement des transactions distribuées afin de s'assurer que toutes les mises à jour présentes sur tous les serveurs sont, soit permanentes, soit supprimées en cas d'erreurs.



Les applications SQL Server peuvent également appeler MS DTC directement afin de démarrer explicitement une transaction distribuée. Un ou plusieurs serveurs exécutant SQL

Server peuvent ensuite être contraints à s'inscrire dans la transaction distribuée et coordonner l'achèvement de cette dernière avec MSDTC.



B) Exécution d'une transaction à l'aide de ADO.NET

Vous pouvez commencer, valider ou annuler une transaction en utilisant les objets Connection et Transaction.
Appelez la méthode BeginTransaction de l'objet Connection pour marquer le début de la transaction. La méthode BeginTransaction retourne une référence à la transaction. Cette référence est affectée aux objets Command inscrits dans la transaction.
Assignez l'objet Transaction à la propriété Transaction de la commande à exécuter. Si la commande est exécutée sur une connexion avec une transaction active et si l'objet Transaction n'a pas été assigné à la propriété Transaction de la commande, une exception est levée.
Exécutez ensuite les commandes requises.
Appelez la méthode Commit de l'objet Transaction pour terminer la transaction ou la méthode Rollback pour l'annuler.
L'exemple de code suivant illustre la logique transactionnelle utilisant ADO.NET avec SQL Server.

```
Dim cn As SqlConnection = New SqlConnection("Data Source=(LOCAL);Initial
Catalog=Northwind;Integrated Security=SSPI;")
cn.Open()

‘Debut de la tansaction
Dim trs As SqlTransaction = cn.BeginTransaction()

' Ajout d’une commande à une transaction.
Dim cmd As SqlCommand = cn.CreateCommand()
cmd.Transaction = trs
```

```
Try
  cmd.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (100,
'Description')"
  cmd.ExecuteNonQuery()
  cmd.CommandText = "Insert into Region (RegionID, RegionDescription) VALUES (101,
'Description')"
  cmd.ExecuteNonQuery()
  cmd.Commit()
  msgbox("les deux enregistrements ont été validés.")
Catch e As Exception
  Try
    trs.Rollback()
  Catch ex As SqlException
    If Not trs.Connection Is Nothing Then
      msgbox("Une exception de type " & ex.GetType().ToString() & _
                  " a été levée au cours de l'annulation de la transaction.")
    End If
  End Try

msgbox("Une exception de type " & e.GetType().ToString() & _
              " a été levée au cours de la validation de la transaction." &
                "Aucun enregsitrement n'a été enregistré.")
Finally
  cn.Close()
End Try
```

Inscription dans une transaction distribuée

L'objet **Connection** s'inscrit automatiquement dans une transaction distribuée existante s'il détermine qu'une transaction est active. L'inscription automatique dans une transaction se produit lorsque la connexion est ouverte. Vous pouvez désactiver l'inscription automatique dans des transactions existantes en spécifiant Enlist=false comme paramètre de chaîne de connexion pour un **SqlConnection** ou OLE DB Services=-7 pour un **OleDbConnection**.
Si l'inscription automatique est désactivée, vous pouvez vous inscrire dans une transaction distribuée existante à l'aide de la méthode **EnlistDistributedTransaction** de l'objet **Connection**. L'inscription dans une transaction distribuée existante garantit que si la transaction vient à être annulée ou validée, les modifications apportées par le code dans la source de données seront elles aussi validées ou annulées.
EnlistDistributedTransaction est particulièrement applicable lors du regroupement d'objets métier. Si un objet métier est regroupé avec une connexion ouverte, l'inscription automatique dans la transaction se produit lorsque cette connexion est ouverte et tirée du pool. Si plusieurs transactions sont effectuées à l'aide de l'objet métier regroupé, la connexion ouverte pour cet objet n'est pas automatiquement inscrite dans les nouvelles transactions lancées. Dans ce cas, vous pouvez désactiver l'inscription automatique dans la transaction pour **Connection** et l'inscrire dans les transactions à l'aide de **EnlistDistributedTransaction**.
EnlistDistributedTransaction accepte un unique argument de type **ITransaction** qui est une référence à la transaction existante. Après avoir appelé **Connection.EnlistDistributedTransaction**, toutes les modifications apportées à la source de données à l'aide de **Connection** sont incluses dans la transaction.

Remarque Le **Connection** doit être ouvert avant l'appel à **EnlistDistributedTransaction**.

ATTENTION **EnlistDistributedTransaction** retourne une **Exception** si le **Connection** a déjà commencé une transaction au moyen de **BeginTransaction**. Toutefois, si la transaction est une transaction locale démarrée dans la source de données (par exemple, l'exécution explicite de l'instruction BEGIN TRANSACTION à l'aide de SqlCommand), **EnlistDistributedTransaction** annule la transaction locale et s'inscrit dans la transaction distribuée existante requise. Vous ne recevez pas d'avis d'annulation de la transaction locale et devez gérer toutes les transactions locales qui n'ont pas été démarrées à l'aide de **BeginTransaction**.