

Université Libanaise

Faculté de Technologie
Génie des Réseaux Informatiques
et Télécommunications



الجامعة اللبنانية

كلية التكنولوجيا

قسم هندسة شبكات المعلوماتية والاتصالات

Architecture Client Serveur

Le Langage ASP.NET - Stockage et Récupération des données avec ADO.NET



Préparé par:

Dr. Youssef ROUMIEH

E-mail : youssef.roumieh@gmail.com

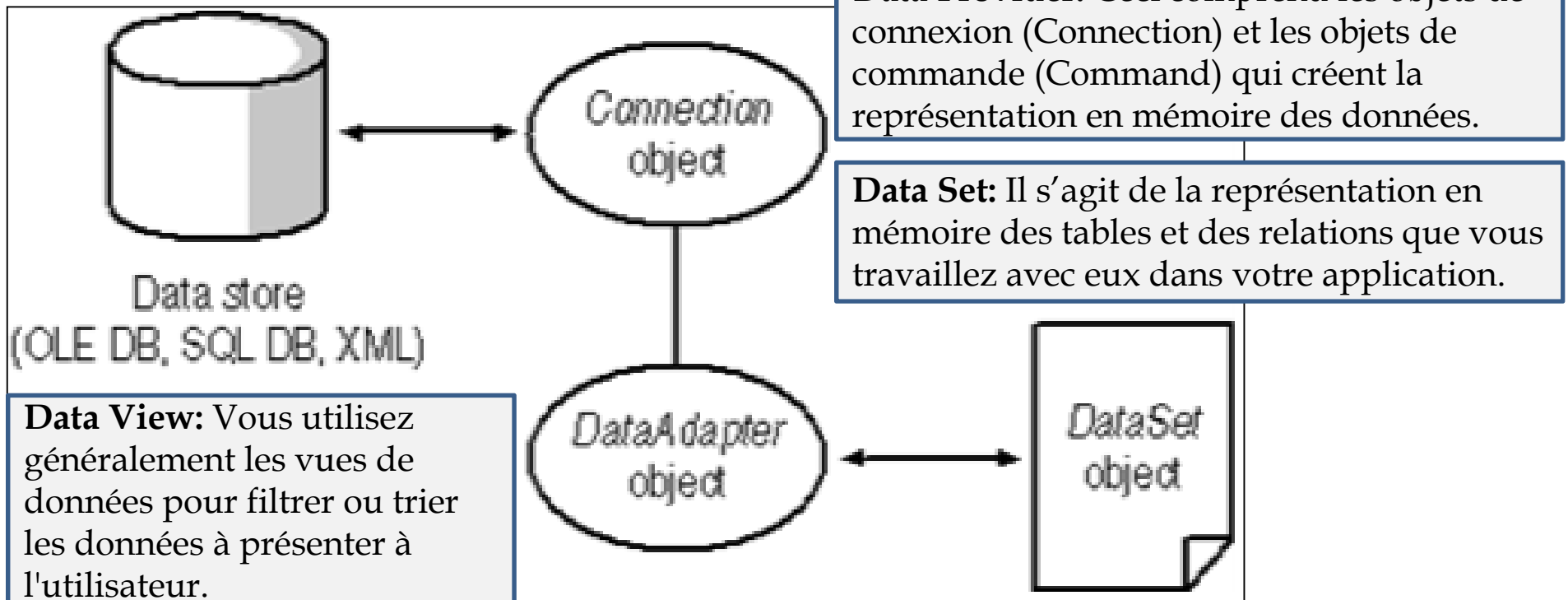
Reference: Web Applications Development with Microsoft .NET Framework 4.

Copyright © 2010 by Tony Northrup and Mike Snell. ISBN: 978-0-7356-2740-6

Les problèmes rencontrés et les oublis constatés dans ce support sont à signaler à youssef.roumieh@gmail.com

Accès aux données avec ADO.NET

- Microsoft ADO.NET : ensemble des outils et des espaces des noms permettant d'accéder aux bases des données.
- **Mode déconnecté**
 - La figure suivante montre le modèle d'objet ADO.NET déconnecté (DataSet est indépendante de Data Store).



- Il existe trois types de connexion de base de données dans ADO.NET :

Namespace	Classe	Pour se connecter à une base de données de type ...	Pour exécuter les requêtes et retourner les résultats
System.Data.OleDb	OleDbConnection	Microsoft Access ou une base de données de tiers, tels que MySQL	OleDbDataAdapter
System.Data.SqlClient	SqlConnection	Microsoft SQL Server	SqlDataAdapter
System.Data.OracleClient	OracleConnection	Oracle	OracleDataAdapter
Note : Utiliser l'espace des noms 'System.Data' pour créer et accéder aux DataSet et DataTable.			

- Dans le code, il faut ajouter les déclarations en utilisant les importations suivantes:

Visual Basic .NET

```
Imports System.Data
' For Microsoft SQL Server database connections.
Imports System.Data.SqlClient
' For Oracle database connections.
Imports System.Data.OracleClient
' For other database connections.
Imports System.Data.OleDb
```

Visual C#

```
using System.Data;
// For Microsoft SQL Server database connections.
using System.Data.SqlClient;
// For Oracle database connections.
using System.Data.OracleClient;
// For other database connections.
using System.Data.OleDb;
```

Exemple

- Etant donné la base de données Vente réalisée sous Access. Elle contient la table :
 - Clients (Numéro, Nom, Crédits)
- Ecrire la page web qui permet d'afficher les informations de tous les clients dans un DataGrid lors de chargement de la page.
- Noter que pour accéder aux données dans une base de données, il faut suivre plusieurs étapes.

Etape 1 : Connexion à la base de données

- Créer un objet de la classe OleDbConnection :

```
OleDbConnection conn = new OleDbConnection() ;
```

- Attribuer une valeur pour ConnectionString selon le SGBD

- Pour Access 2007 :

```
String strConnection = "Provider=Microsoft.ACE.OLEDB.12.0;  
Data Source=      C:\myFolder\myAccess2007file.accdb; Persist  
Security Info=False;" ;
```

```
conn.ConnectionString = strConnection;
```

- ou bien :

```
OleDbConnection conn = new OleDbConnection(  
"Provider=Microsoft.ACE.OLEDB.12.0; Data Source="+  
Server.MapPath("Vente/Vente.accdb") );
```

- **Remarque:**

- vous pouvez trouver les valeurs pour ConnectionString, selon le SGBD à utiliser, à l'adresse suivante : <http://www.connectionstrings.com/>
- Vous pouvez copier sa valeur en utilisant Microsoft Visual Studio.

```

using System.Data;
using System.Data.OleDb;
namespace commande
{
    public partial class Test : System.Web.UI.Page
    {
        OleDbDataAdapter adapter;
        protected void Page_Load(object sender, EventArgs e)
        {
            //(1) create the data conenction
            OleDbConnection conn = new
OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0; " +
                "Data Source=" + Server.MapPath("Vente/Vente.accdb"));
            //(2) Create a data adapter
            adapter = new OleDbDataAdapter("Select * from Clients", conn);
            //(3) Create a data set
            DataSet dsClients = new DataSet();
            //(4) Fill the data set
            adapter.Fill(dsClients, "Clients");
            //(5) Display the table in a data grid using data binding.
            dgClients.DataSource = dsClients.Tables["Clients"].DefaultView;
            dgClients.DataBind();
        }
    }
}

```

- dgClients.DataSource = dsClients.Tables["Clients"].DefaultView:
 - return a DataView, you can use to sort, filter and serach a DataTable
 - DefaultView.RowFilter="Numero < 100"
- dgClients.DataBind();
 - lie une source de donnees au controle serveur.

- Ecrire une méthode qui prend en paramètre un *DataTable* *dt* et qui retourne un numéro du nouveau client. La méthode calcule le numéro de la façon suivante :

- Numéro est égal au nombre de lignes dans *dt* + 1
 - Si le numéro n'existe pas dans *dt*, alors la fonction retourne le numéro.
 - Sinon, il faut parcourir *dt* jusqu'à trouver un numéro déjà supprimé et ensuite retourner ce numéro.

```
// Helper function to get a new, valid row ID.
int GetNewID(DataTable dt)
{
    // Get a new row number.
    int NextID = dt.Rows.Count + 1;
    // If it isn't found in the table, return it.
    if (dt.Rows.Find(NextID) == null) return NextID;
    // Otherwise, check for free IDs between 1 and the row count.
    for (NextID = 1; NextID <= dt.Rows.Count; NextID++)
    {
        // Check if this ID already exists.
        if (dt.Rows.Find(NextID) == null) return NextID;
    }
    // Failed, return zero.
    return 0;
}
```

dt.Rows.Find(NextID) :

- gets the row specified by the primary key value
- returns a *DataRow*

Donner le code de la page web qui permet d'afficher, lors de chargement de la page, tous les clients dans un DataGrid comme suit :

```
protected void Page_Load(object sender, EventArgs e)
{
    //(1) create the data conenction
    OleDbConnection conn = new
OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0; " +
        "Data Source=" + Server.MapPath("Vente/Vente.accdb"));
    //(2) Create a data adapter
    adapter = new OleDbDataAdapter("Select * from Clients", conn);
    //(3) Create a data set
    DataSet dsClients = new DataSet();
    // (4) Fill the data set
    adapter.Fill(dsClients, "Clients");
    // Get the data table
    dtClients = dsClients.Tables["Clients"];
    //Create a primary key on data table.
    dtClients.PrimaryKey = new DataColumn[] { dtClients.Columns["Numero"] };
    // (5) Display the table in a data grid using data binding.
    //dgClients.DataSource = dsClients.Tables["Clients"].DefaultView;
    //dgClients.DataBind();
    dgClients.DataSource = dtClients.DefaultView;
}

public DataColumn[] PrimaryKey {get ; set ;}
```

- Donner le code de la procédure d'événement qui permet lorsqu'un utilisateur clique sur le bouton '**Nouveau**' d'afficher le numéro du nouveau client dans le TextBox txtNumero (appeler la méthode GetNewID écrite dans la question 1 (slide 8)).

```
protected void butNouveau_Click(object sender, EventArgs e)
{
    txtNumero.Text = GetNewID(dtClients).ToString();
}
```

- Donner le code de la procédure d'événement qui permet lorsqu'un utilisateur clique sur le bouton '**Ajouter**' d'ajouter un nouveau client à la table Clients en utilisant la connexion déconnectée.

```
protected void butAdd_Click(object sender, EventArgs e)
{
    // (6) Create a new row object for the Clients
    table.

    DataRow rowNew = dtClients.NewRow();
    // Add data to the columns in the row.
    rowNew["Numero"] = txtNumero.Text;
    rowNew["Nom"] = txtNom.Text ;
    rowNew["Credit"] = txtCredit.Text;
    // Add the row to the data set.
    dtClients.Rows.Add(rowNew);
}
```

```
private void Page_PreRender(object sender, EventArgs e)
{
    // (7) Create insert, delete, and update commands automatically.
    OleDbCommandBuilder cmdContactMgmt = new OleDbCommandBuilder(adapter);

    // (8) Update the database.
    adapter.Update(dtClients);
    // Bind data to DataGrid to update the display.
    dgClients.DataBind();
}
```

- `OleDbCommandBuilder cmdContactMgmt = new OleDbCommandBuilder(adapter);`
Initialise une nouvelle instance de la classe `OleDbCommandBuilder` avec l'objet `OleDbDataAdapter` associé.
- `adapter.Update(dtClients);`
Met à jour les valeurs de la base de données en exécutant les instructions Insert, Update ou Delete respectives pour chaque ligne insérée, mise à jour, ou supprimée dans le `DataTable` spécifié.

`OleDbDataAdapter` ne génère pas automatiquement les instructions requises pour l'harmonisation des modifications apportées à `DataSet` avec la source de données associée. Cependant, vous pouvez créer un objet `OleDbCommandBuilder` pour générer automatiquement des instructions SQL pour la mise à jour de la table simple si vous définissez la propriété `SelectCommand` de `OleDbDataAdapter`.

La méthode `Update` s'ouvre automatiquement la connexion avec la base de données avant d'effectuer les modifications et ferme lorsqu'elle a terminé.

- Donner le code de la procédure d'événement qui permet lorsqu'un utilisateur clique sur le bouton '**Supprimer**' de supprimer un client de la table Clients en utilisant la connexion déconnectée.

```
protected void butDel_Click(object sender, EventArgs e)
{
    int i=0;
    foreach (DataRow row in dtClients.Rows)
    {
        if (((int)row["Numero"]) == Integer.Parse(txtNumero.Text))
            //dtClients.Rows.Remove(row);
            break;
        i++;
    }
    dtClients.Rows.RemoveAt(i);
}
```

Squelette de la page

```
using System.Data;
using System.Data.OleDb;

namespace commande
{
    public partial class UpdateTest : System.Web.UI.Page
    {
        OleDbDataAdapter adapter;
        DataTable dtClients;
        int m_NextID;

        protected void Page_Load(object sender, EventArgs e)
        {
        }

        protected void butAdd_Click(object sender, EventArgs e)
        {
        }

        private void Page_PreRender(object sender, EventArgs e)
        {
        }

        int GetNewID(DataTable dt)
        {
        }

        protected void butDel_Click(object sender, EventArgs e)
        {
        }
    }
}
```

Stockage de plusieurs tables et mise en cache ensembles de données

- Vous pouvez enregistrer les ensembles de données dans les variables Application, Session ou Cache de sorte que vous n'avez pas à les recréer à chaque fois le formulaire est affiché.
- L'objet Cache permet de spécifier une date d'expiration pour les données qu'il contient. Les ensembles de données peuvent être volumineux, et vous ne voulez généralement pas les laisser en mémoire indéfiniment.
- Comme avec l'état de l'Application, tout le code de l'application a accès aux données dans l'objet Cache.

La méthode Add

```
public object Add(string key, object value, CacheDependency dependencies,  
    DateTime absoluteExpiration, TimeSpan slidingExpiration,  
    CacheItemPriority priority, CacheItemRemovedCallback onRemoveCallback  
)
```

▪**dependencies:** [System.Web.Caching.CacheDependency](#)

The file or cache key dependencies for the item. When any dependency changes, the object becomes invalid and is removed from the cache. If there are no dependencies, this parameter contains **null**.

▪**absoluteExpiration:** [System.DateTime](#)

The time at which the added object expires and is removed from the cache. If you are using sliding expiration, the *absoluteExpiration* parameter must be [NoAbsoluteExpiration](#).

▪**slidingExpiration:** [System.TimeSpan](#)

The interval between the time the added object was last accessed and the time at which that object expires. If this value is the equivalent of 20 minutes, the object expires and is removed from the cache 20 minutes after it is last accessed. If you are using absolute expiration, the *slidingExpiration* parameter must be [NoSlidingExpiration](#).

▪**priority:** [System.Web.Caching.CacheItemPriority](#)

The relative cost of the object, as expressed by the [CacheItemPriority](#) enumeration (Low, Below Normal, Normal (default), Above Normal, High, Not removable). The cache uses this value when it evicts objects; objects with a lower cost are removed from the cache before objects with a higher cost.

▪**onRemoveCallback:** [System.Web.Caching.CacheItemRemovedCallback](#)

A delegate that, if provided, is called when an object is removed from the cache. You can use this to notify applications when their objects are deleted from the cache.

Example

```
// Data adapter, combines SQL command and connection string.
SqlDataAdapter adptDB =new SqlDataAdapter("SELECT * FROM Contacts",
    "server=(local);database=Contacts;Trusted_Connection=yes");
// Data set to contain two data tables
public DataSet dsBoth = new DataSet("Both");

private void Page_Load(object sender, System.EventArgs e)
{
    // Create Cache if this is the first time the page is displayed
    // or if the Cache object doesn't yet exist.
    if ((!IsPostBack) || (Cache.Get("dsBoth") == null))
    {
        // Create the Contacts table.
        DataTable Contacts = new DataTable("Contacts");
        adptDB.Fill(Contacts);

        // Change the adapter's SELECT command.
        adptDB.SelectCommand.CommandText = "SELECT * FROM Calls";
        // Create the Calls table.
        DataTable Calls = new DataTable("Calls");
        adptDB.Fill(Calls);

        // Continue on the next slide
    }
}
```

Example

```
private void Page_Load(object sender, System.EventArgs e)
{
    // ...
    // Add both tables to a single dataset.
    dsBoth.Tables.Add(Calls);
    dsBoth.Tables.Add(Contacts);

    // Cache data set for 20 minutes.
    Cache.Add("dsBoth", dsBoth, null, DateTime.MaxValue,
        System.TimeSpan.FromMinutes(20),
        System.Web.Caching.CacheItemPriority.Default, null);
    // Bind the drop-down list to display data.
    drpContacts.DataBind();
}
else
    // If this is post-back get the Cached data set.
    dsBoth = (DataSet) Cache["dsBoth"];
}
```

Exemple

```
private void drpContacts_SelectedIndexChanged(object sender,
    System.EventArgs e)
{
    // Set a filter on the view of the Calls table.
    dsBoth.Tables["Calls"].DefaultView.RowFilter = "ContactID=" +
    drpContacts.SelectedItem.Value.ToString();
    // Bind to the data grid.
    grdCalls.DataBind();
}
```

```
dsBoth.Tables["Calls"].DefaultView.RowFilter = "ContactID=" +
    drpContacts.SelectedItem.Value.ToString();
to filter which rows are viewed in the DataView.
```

Afficher les éléments de données dans d'autres contrôles de liste

- Le code HTML suivant montre les paramètres de propriété pour un contrôle DropDownList qui est lié à la DataSet Contacts:

```
<asp:DropDownList id="drpContacts" runat="server"
Width="384px" Height="22px" DataSource="<%# dsContacts1 %>"
DataTextField='LastName' DataValueField="ContactID"></asp:DropDownList>
```

- Le code suivant remplit la DataSet et lie les données au contrôle DropDownList:

```

using System.Data.OleDb;

namespace commande
{
    public partial class DropDownListTest : System.Web.UI.Page
    {
        OleDbDataAdapter adapter;
        protected void Page_Load(object sender, EventArgs e)
        {
            //(1) create the data conenction
            OleDbConnection conn = new
OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0; " +
                "Data Source=" + Server.MapPath("Vente/Vente.accdb"));
            //(2) Create a data adapter
            adapter = new OleDbDataAdapter("Select * from Clients", conn);
            //(3) Create a data set
            DataSet dsClients = new DataSet();
            // (4) Fill the data set
            adapter.Fill(dsClients, "Clients");
            // Bind the data to the DataList control.
            ddlClients.DataSource = dsClients;
            ddlClients.DataTextField = "Nom";
            ddlClients.DataValueField = "Numero";

            ddlClients.DataBind();
        }
    }
}

```

Remarque :

La limitation ici est que vous pouvez inclure une seule valeur pour les propriétés du contrôle de liste DataText et DataValue. Si vous voulez inclure les nom et prénom du contact dans le contrôle DropDownList, vous devez le faire à partir du code.

Exemple

```
Private void Page_Load(object sender, System.EventArgs e)
{
    // Run first time page is displayed.
    if (!IsPostBack)
    {
        //(1) create the data conenction
        OleDbConnection conn = new
OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0; " +
        "Data Source=" + Server.MapPath("Vente/Vente.accdb"));
        //(2) Create a data adapter
        adapter = new OleDbDataAdapter("Select * from Clients", conn);
        //(3) Create a data set
        DataSet dsClients = new DataSet();
        //(4) Fill the data set
        adapter.Fill(dsClients, "Clients");

        // continue on the next slide
    }
}
```

Exemple

```
Private void Page_Load(object sender, System.EventArgs e)
{
    // ...

    // For each row in the table...
    foreach (DataRow drowItem in
dsClients.Tables["Clients"].Rows)
    {
        // Create a new list item.
        ListItem lstNew = new ListItem();
        lstNew.Text = drowItem["Nom"] + " " +
            drowItem["Numero"];
        lstNew.Value = drowItem["Numero"].ToString();
        // Add the list item to the drop-down list.
        ddlClients.Items.Add(lstNew);
    }
}
```

Autre methode:

```
Select Numero, Nom & ' ' & Prenom as 'name' from Clients.
```

Sélection d'enregistrements spécifiques

- Le contrôle DropDownList créé dans la section précédente est utile pour sélectionner des éléments de la table Clients dans la base de données Vente.
- Parce que Numéro est une clé primaire dans la base de données, vous pouvez l'utiliser pour sélectionner les enregistrements de commande pour un client spécifique.


```
protected void ddlClients_SelectedIndexChanged(object sender, EventArgs e)
{
    DataSet dsOrder = new DataSet();

    //(1) create the data conenction
    OleDbConnection conn = new
OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0; " +
        "Data Source=" + Server.MapPath("Vente/Vente.accdb"));
    //(2) Create a data adapter
    adapter = new OleDbDataAdapter("Select * from Clients", conn);

    //When a client is selected, display the client's information.
    adapter.SelectCommand.CommandText = "Select * from Commandes" +
        " where Client = " + ddlClients.SelectedItem.Value.ToString();

    adapter.Fill(dsOrder, "Order");
    if (dsOrder.Tables["Order"].Rows.Count > 0)
    {
        //Display the result in a data grid.
        dgClients.DataSource = dsOrder;
        dgClients.DataBind();
        dgClients.Visible = true;
        lblMsg.Text = "";
    }
    else
    {
        dgClients.Visible = false;
        lblMsg.Text = ddlClients.SelectedItem.Text + "has no orders.";
    }
}
```

Exécuter des commandes directement sur une base de données (en mode connecté)

- L'objet de connexion de base de données fournit les trois méthodes suivantes:
 - **ExecuteScalar:** Effectue les commandes de requête qui renvoient une valeur unique, comme compter le nombre d'enregistrements dans une table.
 - **ExecuteNonQuery:** Effectue les commandes qui modifient la base de données, mais ne renvoient pas une valeur spécifique, y compris l'ajout et la suppression d'éléments dans une base de données. La méthode ExecuteNonQuery renvoie le nombre de lignes affectées par la commande.
 - **ExecuteReader:** Lit les enregistrements séquentiellement à partir de la base de données.
- Pour utiliser ces méthodes, suivez ces étapes:
 - Créez une connexion à la base de données.
 - Ouvrez la connexion.
 - Créez un objet de Command contenant la commande SQL ou une procédure stockée à exécuter.
 - Exécutez la méthode sur l'objet de Command.
 - Fermez la connexion de base de données.
 - **Remarque :** Vous devriez toujours utiliser la gestion des exceptions pour s'assurer que la connexion de base de données est fermée si la commande réussit ou non.

Renvoi d'une valeur à partir d'une base de données

- Le code suivant exécute une fonction SQL MAX pour récupérer le plus grand nombre utilisé pour ContactID et utilise ensuite cette valeur comme la valeur de départ pour ajouter une nouvelle ligne à la table Contacts:

```
SqlConnection connContacts = new SqlConnection("integrated security=SSPI;" +
    "data source=(local);initial catalog=Contacts");
private void butAdd_Click(object sender, System.EventArgs e){
    try    {
        // Create a SQL command to get a unique ContactID.
        SqlCommand cmdNewID =new SqlCommand("SELECT MAX(ContactID)"+
            "FROM Contacts", connContacts);
        // Open the database connection.
        connContacts.Open();
        // Execute the SQL command.
        int intNextID  = (int)cmdNewID.ExecuteScalar() + 1;
        // Create a command to add a new row.
        SqlCommand cmdAddRow = new SqlCommand(String.Format("INSERT ` +
            "INTO Contacts(ContactID, FirstName, LastName, WorkPhone) " +
            "VALUES({0}, '{1}', '{2}', '{3}')" , intNextID,
            txtFirstName.Text, txtLastName.Text,
            txtPhone.Text), connContacts);
        //..continue on the next slide
    }
```

Renvoi d'une valeur à partir d'une base de données

- Le code suivant exécute une fonction SQL MAX pour récupérer le plus grand nombre utilisé pour ContactID et utilise ensuite cette valeur comme la valeur de départ pour ajouter une nouvelle ligne à la table Contacts:

```
// Execute the command.
if (cmdAddRow.ExecuteNonQuery() > 0)      {
    lblMsg.Text = "Record added.";
    // Database changed, so refresh list.
    RefreshList();
    // Clear text boxes.
    ClearText();
}
else{
    lblMsg.Text = "Couldn't add record.";
}
}
catch (Exception ex)    {
    lblMsg.Text = "Couldn't access database due to this error: "
        + ex.Message;
}
finally    {
    // Close connection
    connContacts.Close();
}
}
```

Changement des enregistrements directement dans une base de données

- Le code suivant utilise ExecuteNonQuery pour supprimer une ligne directement à partir de la base de données:

```
SqlConnection connContacts = new SqlConnection("integrated security=SSPI;" +
    "data source=(local);initial catalog=Contacts");

private void butDelete_Click(object sender, System.EventArgs e)
{
    try
    {
        // Create a SQL command to delete record.
        SqlCommand sqlDelete = new SqlCommand("DELETE FROM Contacts " +
            "WHERE ContactID=" + lblContactID.Text, connContacts);
        // Open the database connection
        connContacts.Open();
        // Execute the command.
        if (sqlDelete.ExecuteNonQuery() > 0)
        {
            lblMsg.Text = "Record deleted.";
            // Database changed, so refresh list.
            RefreshList();
            // Clear text boxes
            ClearText();
        }
    }
}
```

```
private void butDelete_Click(object sender, System.EventArgs e)
{
    //...
    else
    {
        lblMsg.Text = "Couldn't find record to delete.";
    }
}
catch (Exception ex)
{
    lblMsg.Text = "Couldn't access database due to this error: " +
        ex.Message;
}
finally
{
    // Close connection
    connContacts.Close();
}
}
```

- **Remarque:**

- La méthode `ExecuteNonQuery` agit directement sur la connexion de base de données connexion.
- Il ne passe pas par un adaptateur de données ou d'un ensemble de données. Si vous apportez des modifications à une table dans la base de données par le biais de `ExecuteNonQuery`, vous devez mettre à jour les ensembles de données touchées par ces changements en appelant la méthode `Fill` sur l'adaptateur de données.

Récupérer des enregistrements directement à partir de la base de données

- Chaque enregistrement est retourné comme un objet de DataReader, qui est une sorte de version en lecture seule d'un ensemble de données.
- Les DataSets vous permettent d'obtenir des enregistrements dans ne importe quel ordre et, plus important, vous permettent d'écrire les modifications dans la base de données.
- Le code suivant obtient tous les contacts et affiche leurs noms dans une liste déroulante en utilisant un objet DataReader et la commande ExecuteReader:

```
SqlConnection connContacts = new SqlConnection("integrated security=SSPI;"
+      "data source=(local);initial catalog=Contacts");

private void Page_Load(object sender, System.EventArgs e)
{
    // Get values for drop-down list the first time
    // the page is changed.
    if (!IsPostBack)
    {
        RefreshList();
    }
}
```

```

private void RefreshList()
{
    try
    {
        // Clear the list
        drpContacts.Items.Clear();
        // Create SQL command to get table.
        SqlCommand cmdGetContacts = new SqlCommand("Select * " +
            "From Contacts", connContacts);
        // Open database connection if it is closed.
        if (connContacts.State == ConnectionState.Closed)
            connContacts.Open();
        // Execute command.
        SqlDataReader readContacts = cmdGetContacts.ExecuteReader();
        // Read the names and contact IDs into a drop-down list
        while (readContacts.Read())
        {
            // Create a new list item.
            ListItem NewItem = new ListItem();
            // Set the item's values.
            NewItem.Text = readContacts.GetString(1) + " "
                + readContacts.GetString(2);
            NewItem.Value = readContacts.GetValue(0).ToString();
            // Add the new item to the list.
            drpContacts.Items.Add(NewItem);
        }
    }
    //...
}

```



```
private void RefreshList()
{
    //...
    catch (Exception ex)
    {
        lblMsg.Text = "Couldn't access data due to this error: " +
            ex.Message;
    }
    finally
    {
        // Always close the connection.
        connContacts.Close();
    }
}
```

- **Remarque:**

- L'objet de DataReader est un ensemble d'enregistrements en lecture vers l'avant, de sorte que la méthode Read lit chaque ligne suivante jusqu'à ce qu'elle atteigne la fin de l'ensemble d'enregistrements.
- Un objet Reader verrouille la connexion de base de données pendant son exécution, de sorte que vous devriez appeler la méthode Close de l'objet Reader lorsque vous avez terminé d'obtenir les enregistrements.

Exécution de procédures stockées

- Une astuce de performance bien connue lorsque vous travaillez avec des bases de données est de déplacer des tâches fréquemment utilisées dans des procédures stockées.
 - Les procédures stockées exécutent dans le contexte du gestionnaire de base de données et peuvent donc être optimisés pour une performance idéale.
- Utilisez la méthode `ExecuteScalar`, `ExecuteNonQuery` ou `ExecuteReader` pour exécuter des procédures stockées.
- Par exemple, le code suivant exécute une procédure stockée qui renvoie les dix produits les plus coûteux dans la base de données Northwind Traders.

```
private void butExecute_Click(object sender, System.EventArgs e)
{
    // Create a connection for NorthWind Traders database.
    SqlConnection connNWind = new SqlConnection("integrated " +
        "security=SSPI;data source=(local);initial catalog=Northwind");
    // Create a command object to execute.
    SqlCommand cmdTopTen = new SqlCommand("Ten Most Expensive Products",
        connNWind);
    // Set the command properties.
    cmdTopTen.CommandType = CommandType.StoredProcedure;
    // Create a data reader object to get the results.
    SqlDataReader drdTopTen;
    // Open the connection.
    connNWind.Open();
    try
    {
        // Execute the stored procedure.
        drdTopTen = cmdTopTen.ExecuteReader();
        // Display a header.
        litData.Text = "<h3>Ten Most Expensive Products:</h3>";
        // ..
    }
}
```

```
private void butExecute_Click(object sender, System.EventArgs e)
{
    //..
    // Display the results on the page.
    while (drdTopTen.Read())
    {
        // Create an array to receive data.
        object[] items = {"", "", "", "", "", ""};
        // If the row contains items.
        if (drdTopTen.GetValues(items) > 0)
        {
            // Add each row item to the literal control.
            foreach(object item in items)
                litData.Text += item.ToString() + " ";
            // Add a break between rows.
            litData.Text += "<br>";
        }
    }
}
catch (Exception ex)
{
    litData.Text = "The following error occurred: <br>";
    litData.Text += ex.Message;
}
finally
{
    // Close the connection.
    connNWind.Close();
}
}
```

Architecture Client Serveur - TD7

- On considère la base de données 'VENTE' suivante :
 - **Clients** (Numero, Nom, Crédit) ;
 - **Articles** (Code, Description, Prix Unitaire)
 - **Commandes** (Numero, DateCommande, Client*) ;
 - **DetailsCommandes** (Commande*, Article*, Quantité, Prix Facturé)
- Donner les codes des pages ASP.NET permettant de réaliser les schémas suivants. Ces pages affichent les données à partir de la base VENTE : ACCESS. Cet affichage doit être valable quel que soit le client, et quel que soit le nombre de ses commandes.

Page : Clients

No Client: 124 ▼

AfficherCommandes Liste

AfficherCommandes Href

Page : ListeCommandes

Le Client: 124

a passé les commandes suivantes : 12500 ▼

DetailCommande

Page : Transfer

Page : DetailCommande

Detail de la commande : 12500

Article	Quantité	Prix Facturé
BT04	1	149.99
CB03	2	310
BA74	15	25
AX12	10	23
BZ66	2	405

Page : HrefCommandes

Le Client: 124

a passé les commandes suivantes :

Le 02/09/2003 Commande No: [12489](#)

Le 05/09/2003 Commande No: [12500](#)

Architecture Client Serveur - TD8

- On considère la base de données 'VENTE' suivante :
 - **Clients** (Numero, Nom, Crédit) ;
 - **Articles** (Code, Description, Prix Unitaire)
 - **Commandes** (Numero, DateCommande, Client*) ;
 - **DetailsCommandes** (Commande*, Article*, Quantité, Prix Facturé)
- Donner les codes des pages ASP.NET permettant de réaliser les schémas suivants. Ces pages affichent les données à partir de la base VENTE : ACCESS. Cet affichage doit être valable quel que soit le client, et quel que soit le nombre de ses commandes.

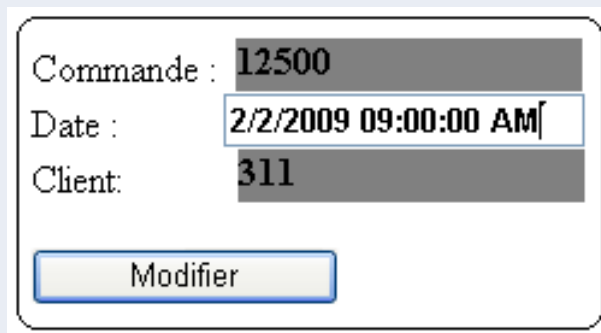
Modifier Details

Transfer.aspx

```
<html>      <head>      <title>Default.html</title>
</head>
<body>
    <form method=post action=Transfer.aspx>
        No Commande:
        <input name="NoCmd" type="text" />
        <br />
        <input      name="action"      value="Modifier
Date"
                        type="submit" />
        &nbsp;
        <input      name="action"      value="Modifier
Details"
                        type="submit" />
    </form>
</body> </html>
```

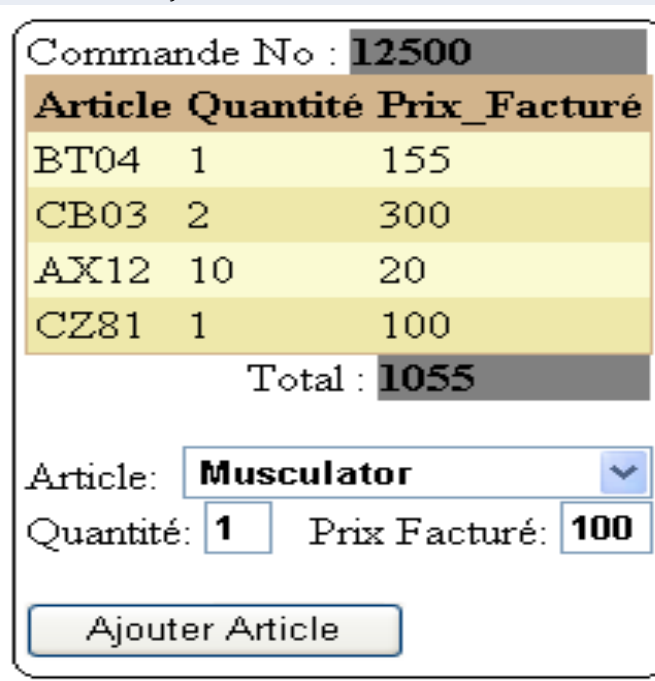
Transfer.aspx: Cette page n'affiche rien, mais elle permet de rediriger l'utilisateur selon le bouton cliqué dans la page Default.html, vers la page ModifierDateCommande.aspx s'il a cliqué sur le bouton Modifier Date ou vers la page ModifierDetailsCommande.aspx s'il a cliqué sur le bouton Modifier Details.

ModifierDateCommande.aspx: Cette page permet d'afficher le numéro, la date et le client concernant la commande saisie dans Default.html. Quand on clique sur le bouton Modifier, cette page permet de mettre à jour la date de la commande dans la base de données et de rediriger l'utilisateur vers la page ModifierDetailsCommande.aspx.



Commande : 12500
Date : 2/2/2009 09:00:00 AM
Client: 311
Modifier

ModifierDetailsCommande.aspx: Cette page permet d'afficher le numéro, les détails et le total concernant la commande saisie dans Default.html ou modifiée dans ModifierDateCommande.aspx. Elle permet aussi d'ajouter un nouvel article à la commande affichée quand on clique sur le bouton Ajouter Article. Un même article ne peut pas être ajouté plus qu'une fois. Le total de la commande varie avec les données ajoutées.



Commande No : 12500

Article	Quantité	Prix_Facturé
BT04	1	155
CB03	2	300
AX12	10	20
CZ81	1	100

Total : 1055

Article: Musculator
Quantité: 1 Prix Facturé: 100
Ajouter Article