# Chapitre 3 bis : le middleware

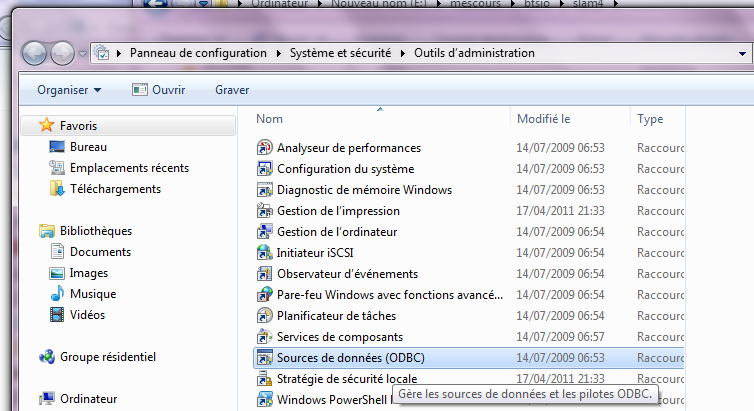
## ODBC : Open Database Connectivity

ODBC est une interface de programmation **rendant n’importe quel logiciel capable** d’interroger une base de données. Elle nécessite tout de même l’installation sur le poste client **de pilotes spécifiques à chaque SGBD** auquel on souhaite se connecter. Le pilote se charge de traduire la demande du logiciel sous une forme compréhensible par le SGBD. Principal intérêt : la requête est la même, quelle que soit la base de données visée. Il n’est donc plus nécessaire de modifier les programmes si cette dernière change, il suffit d’installer un nouveau pilote. Mais cette compatibilité a un prix en terme de rapidité, surtout si les transferts sont nombreux entre les logiciels et le SGBD. Une évolution a donc vu le jour pour optimiser les communications avec les serveurs : ODBC Direct. A noter aussi que, bien qu’initialement prévu pour la connexion à des serveurs SQL, ODBC dispose de nombreux pilotes pour lire tout type de base de données.

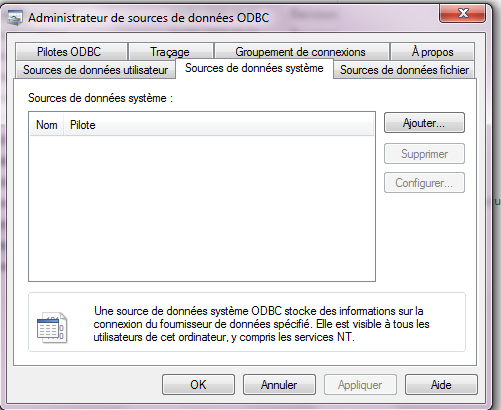
### Configuration d’ODBC pour MYSQL

Cet exemple montre une configuration d’OBC pour une base Mysql

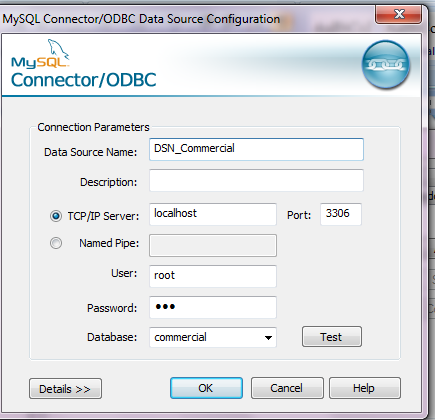
Ouvrez le panneau de configuration



Sélectionnez Sources de données (ODBC) et cliquez sur l’onglet Sources de données système ( accessible pour tous les utilisateurs)



Cliquez sur le bouton Ajouter



Choisissez le driver ODBC pour Mysql .

L’application écrite en C# va utiliser le nom logique qui désigne la DSN.

## Développement en couches

Une application client serveur se doit de suivre une structure qui met en œuvre les couches suivantes :

* La couche présentation : elle représente la partie front office de l’application, c’est elle qui est en relation directe avec l’utilisateur. Sa responsabilité s’arrête au contrôle quand cela est possible de la saisie des données.
* La couche persistance représente l’accès aux données. Elle possède deux classes :
  + Connexion : classe qui va servir à la connexion avec la base de données.
  + DbInterface : classe qui possède les méthodes qui vont dialoguer avec la base de données soit pour faire des lectures ou des écritures.
* La couche metier posséde les classes qui sont en relation avec la base de données. Chaque table donne naissance à une classe métier. Dans chaque classe, on écrit le traitement spécifique. Cette couche est en relation avec la couche présentation et la couche persistance.
* La couche utilitaires représente les outils que l’application peut être amenée à utiliser. En général elle comprend les classes et les structures de données nécessaires à la gestion des erreurs. Dans notre exemple, nous trouverons la classe MonException qui génère une exception personnalisée. Cette couche est accessible depuis toutes les autres.

## Développement en couches

Cette application suit un développement en couches

Couche Présentation

Couche Utilitaires

Couche Métier

Couche Persistance

## Application C# : Connexion à Mysql via ODBC

A présent nous pouvons créer une application de type Client/serveur qui va se connecter à ODBC en utilisant la DSN définie çi-dessus.

## Structure de l’application

Nous allons structurer notre application en différents packages dont l’utilité vous est présentée dans le tableau suivant :

|  |  |  |
| --- | --- | --- |
| **Bibliothèque de classes** | **Utilité** | **Classes** |
| Persistance | Permet de dialoguer avec la base de données | Connexion  DbInterface |
| Metier | Classes métier | Clientel  Vendeur  Commande  …. |
| **MesErreurs** | Accessible depuis toutes les couches | Gère les erreurs  MonException |
| Presentation | Fenêtres de l’application | Winforms de l’application |

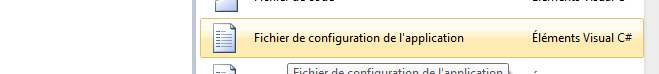
## Couche Persistance

Cette couche d’infrastructure gère la liaison avec la base de données. Elle comprend les classes suivantes :

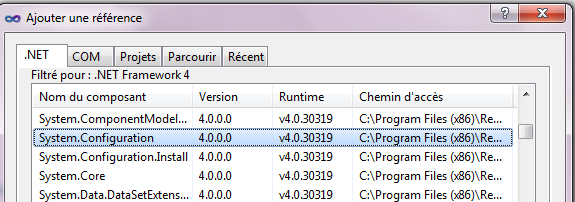
|  |  |
| --- | --- |
| * Connexion * DbInterface |  |

La classe Connexion va établir une connexion avec la base de données en lisant un fichier Xml nommé App.Config

Il faut ajouter un composant à l’application



Puis vous devez ajouter la référence à votre projet



Ce fichier contient les éléments pour se connecter :

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<appSettings>

<add key="bddCourante" value="commercial"/>

<add key="commercial"

value="SERVER=localhost;DSN=DSN\_COMMERCIAL;UID=userSIO ;PWD=sio;"/>

</appSettings>

</configuration>

La premire clé **bddCourante** donne le nom de la base

La seconde clé **commercial\_mysql** donne la chaîne de connexion : serveur, nom de base, utilisateur, mot de passe.

**Lecture des clés**

string bddCourante = ConfigurationManager.AppSettings["bddCourante"];

string strConnexion = ConfigurationManager.AppSettings[bddCourante];

La classe Connexion **va utiliser un singleton pour être sûr de n’avoir qu’une instance de la base.** Elle sera appelée par la classe DbInterface qui contient les requêtes SQL à exécuter de la manière suivante :

cnx = Connexion.getInstance().getConnexion();

On vérifie s’il existe une instance puis on initialise la connexion avec les paramètres du fichier App.Config

#### Travail à faire

#### Créez le fichier App.ccnfig à la racine de votre projet :

### Classe Connexion

Cette classe possède les méthodes pour établir une connexion avec la base de données. Pour être sûr de n’avoir qu’une seule instance de cette classe, on crée :

* un singleton getInstance()
* un constructeur privé

**Les méthodes sont statiques, elles sont appelées à travers la classe.**

#### Code de la classe Connexion

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Sstem.Data.Odbc;

using System.Configuration;

using Utilitaires;

namespace Persistance

{

public class Connexion

{

private static OdbcConnection macnx;

/// <summary>

///

/// </summary>

///

/// Impossible de créer un objet connexion

///

private Connexion ()

{

}

private static Connexion instance;

public OdbcConnection getConnexion()

{

try

{

string bddCourante = ConfigurationManager.AppSettings["bddCourante"];

string strConnexion = ConfigurationManager.AppSettings[bddCourante];

macnx = new MySqlConnection(strConnexion);

macnx.Open();

return macnx;

}

catch (OdbclException err)

{

throw new MonException("", "Erreur d'acces à la base de Gestion des frais", err.Message);

}

}

/// <summary>

/// On crée le singleton

/// </summary>

public static Connexion getInstance()

{

if (instance == null)

instance = new Connexion();

return instance;

}

public static void closeConnexion()

{

macnx.Close();

}

}

}

#### Travail à faire

#### Ajoutez à la bibliothèque de classe Persitance : la classe Connexion ( )

### Couche Persistance : classe DbInterface

Cette classe contient la méthode de lecture qui est **statique**, elle est appelée à travers la classe.

Voici à tire d’exemple la méthode de lecture :

using System;

using System.Collections.Generic;

using System.Text;

using System.Data;

using MesErreurs;

using System.Data.Odbc;

namespace Persistance

{

public class DbInterface

{

private static DbInterface instance;

private DbInterface()

{ }

/// <summary>

/// On crée le singleton

/// </summary>

public static DbInterface getInstance()

{

if (instance == null)

instance = new DbInterface();

return instance;

}

public static DataTable Lire\_Donnees(string requete)

{

try

{

// Construire la requète

OdbcCommand cmd = new OdbcCommand();

cmd.Connection = Connexion.getInstance().getConnexion();

cmd.CommandText = requete;

OdbcDataAdapter da = new OdbcDataAdapter();

da.SelectCommand = cmd;

// Construire le DataSet

DataSet ds = new DataSet();

// Remplir le Dataset

da.Fill(ds, "resultat");

// Récupérer la table client

DataTable dt = ds.Tables["resultat"];

return (dt);

}

catch (OdbcException uneException)

{

new MonException(uneException.Message, "Lecture", "SQL");

return null;

}

}

}

}

#### Travail à faire

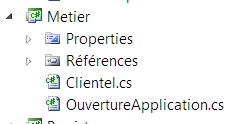
#### Ajoutez à la bibliothèque de classe Persitance : la classe DbInterface ( )Lecture retourne un DataTable en réponse à une requête SQL. Elle s’appelle à travers une exception.

## Couche metier

Cette couche contient les classes issues de la base de données. Pour chaque classe, on écrit les traitements spécifiques. Voici le code d’une classe métier Visiteur (qui n’appartient pas au projet). Chaque classe de la couche métier suit la même structure.

#### Travail à faire

#### Ajoutez à la bibliothèque de classe Metier : la classe Clientel,



## Couche MesErreurs

Cette couche offre les méthodes de la gestion des erreurs ;; voici le code de cette classe  :

using System;

using System.Collections.Generic;

using System.Text;

using System.Diagnostics;

using System.Configuration;

using System.Web;

using System.IO;

namespace MesErreurs

{

public class MonException : Exception

{

private string \_utilisateur, \_application, \_systeme;

private string \_support = "Si l'erreur persiste, relevez les messages ci-dessus et prenez contact avec le support technique.";

public MonException()

{

\_utilisateur = \_application = \_systeme = "";

}

public string MsgUtilisateur

{

get { return \_utilisateur; }

set { \_utilisateur = value + "\r\n"; }

}

public string MsgApplication

{

get { return \_application; }

set { \_application = "Origine de l'erreur : " + value + "\r\n"; }

}

public string MsgSysteme

{

get { return \_systeme; }

set { \_systeme = "Erreur système : " + value + "\r\n"; }

}

public MonException(string u, string a, string s)

{

\_utilisateur = \_application = \_systeme = "";

if (u != "")

\_utilisateur = u + "\r\n";

if (a != "")

\_application = "Origine de l'erreur : " + a + "\r\n";

if (s != "")

\_systeme = "Erreur système : " + s + "\r\n";

public string MessageUtilisateur()

{

return (\_utilisateur);

}

public string MessageApplication()

{

return (\_application);

}

public string MessageSysteme()

{

return (\_systeme);

}

public string MessageSupport()

{

return (\_support);

}

public string Messages()

{

if (\_systeme == "")

\_support = "";

return (\_utilisateur + \_systeme + \_support);

}

}

}

#### Travail à faire

#### Ajoutez à la bibliothèque de classe Utilitaires : la classe MonException

#### ,

### Ouverture de l’application

Au lancement de l’application, il faut tester la connexion pour savoir si la base est en ligne ou non. On va donc créer une classe nommée OuvertureApplication dans le package Metier, qui sera chargée de ce test.

Voici son code :

using System.Linq;

using System.Text;

using System.Data;

using MySql.Data.MySqlClient;

using System.Configuration;

using Persistance;

using Utilitaires;

namespace Metier

{

public class OuvertureApplication

{

/// <summary>

/// Tentative de connexion à la base

/// </summary>

/// <returns></returns>

static public bool getOuverture()

{

try

{

Connexion macnx = Connexion.getInstance();

MySqlConnection mysqlcnx = macnx.getConnexion();

return true;

}

catch (MonException excep)

{

throw excep;

}

}

}

}

## Fenêtre principale : test de la connexion

A l’ouverture de l’application , il faut tester si la base est en ligne en appelant la méthode getOuverture ().

On peut donc intervenir sur l’événement \_load qui se déclenche au chargement de la fenêtre :

Ce code est relativement simple, il suffit d’appeler la méthode de connexion et de tester le retour de cette fonction pour activer ou non les items du menu.

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using Metier;

namespace CommercialODBC

{

public partial class fAccueil : Form

{

public fAccueil()

{

InitializeComponent();

}

private void fAccueil\_Load(object sender, EventArgs e)

{

quitterToolStripMenuItem.Enabled = true;

if (OuvertureApplication.getOuverture() )

{

interrogerToolStripMenuItem.Enabled = true;

lbl\_etat.Text = "Etat : en ligne";

}

else

{

interrogerToolStripMenuItem.Enabled = false;

lbl\_etat.Text = "Etat : hors ligne";

}

}

private void ficheClientToolStripMenuItem\_Click(object sender, EventArgs e)

{

fClient fc;

fc = new fClient();

fc.ShowDialog();

}

private void quitterToolStripMenuItem\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

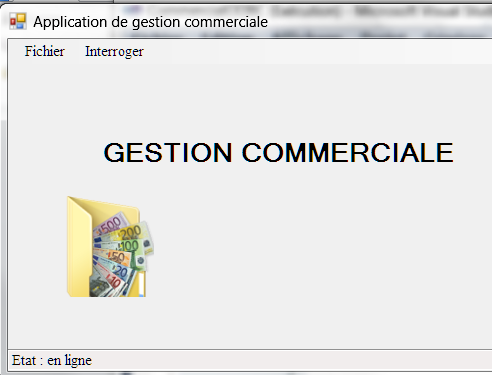
#### Travail à faire

#### Ajoutez à la bibliothèque de classe :

#### - OuvertureApplication

#### - FermetureApplication

#### Testez votre application pour afficher l’état de la base : en ligne ou non.



## Information sur un client

Vous allons à présent écrire des requêtes SQL . On désire connaître la société, le nom et la ville d’un client dont le numéro est choisi dans une liste déroulante.

### Démarche de construction : classe Clientel

Cette classe correspond à la table clientel. Elle appartient à la couche métier. Elle possède des accesseurs, des constructeurs et des traitements spécifiques sur les données :

* Lecture de tous les numéros client
* Lecture des informations d’un client

#### Code de cette classe

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Data;

using Persistance;

using MesErreurs;

//Modele de developpement en couche - Modele 3-tiers

//Cette classe appartient a la couche metier. La couche metier sert a faire communiquer la couche présentation

//et la couche d'accés aux données (data acess layer ou couche persistante).

//Chacune des couches ne peut communiquer qu'avec la couche directement adjacente.

//Ainsi les couches présentation et persistante ne doivent pas communiquer ensemble

//Une seule exception est faite en utilisant la couche persistante pour la connexion dans la fenetre principale

//de la couche présentation.

namespace Metier

{

public class Clientel

{

//Definition des attributs

private String noClient;

private String societe;

private String nomCl;

private String prenomCl;

private String adresseCl;

private String villeCl;

private String codePostCl;

//Definition des properties

public String Societe

{

get { return societe; }

set { societe = value; }

}

public String NomCl

{

get { return nomCl; }

set { nomCl = value; }

}

public String PrenomCl

{

get { return prenomCl; }

set { prenomCl = value; }

}

public String AdresseCl

{

get { return adresseCl; }

set { adresseCl = value; }

}

public String VilleCl

{

get { return villeCl; }

set { villeCl = value; }

}

public String CodePostCl

{

get { return codePostCl; }

set { codePostCl = value; }

}

//Constructeurs

public Clientel()

{

noClient = "";

societe = "";

nomCl = "";

prenomCl = "";

adresseCl = "";

villeCl = "";

codePostCl = "";

}

public Clientel(string no, string soc, string nom, string prenom, string adresse, string ville, String codePostal)

{

noClient = no;

societe = soc;

nomCl = nom;

prenomCl = prenom;

adresseCl = adresse;

villeCl = ville;

codePostCl = codePostal;

}

//Traitement de la classe Clientel

/// <summary>

/// Lire un utilisateur sur son ID

/// </summary>

/// <param name="id\_utilisateur">N° de l'utilisateur à lire</param>

public List<String> LectureNoClient()

{

List<String> mesNumeros = new List<String>();

DataTable dt;

try

{

String mysql = "SELECT DISTINCT NO\_CLIENT FROM CLIENTEL ORDER BY NO\_CLIENT";

dt = DbInterface.Lire\_Donnees(mysql);

foreach (DataRow dataRow in dt.Rows)

{

mesNumeros.Add((dataRow[0]).ToString());

}

return mesNumeros;

}

catch (MonException e)

{

throw new MonException("Erreur sur lecture du client.", "Utilisateur.lectureClient()", e.Message);

}

}

public Clientel RechercheUnClient(String numCli)

{

String mysql;

DataTable dt;

try

{

mysql = "SELECT SOCIETE, NOM\_CL, PRENOM\_CL,";

mysql += "ADRESSE\_CL, VILLE\_CL, CODE\_POST\_CL ";

mysql += "FROM CLIENTEL WHERE NO\_CLIENT=':numero'";

// On se protège de l’injection sql

mysql = mysql.Replace(":numero", numCli);

// On appelle la couche Persistance

dt = DbInterface.Lire\_Donnees(mysql);

if (dt.IsInitialized)

{

DataRow dataRow = dt.Rows[0];

this.noClient = numCli;

this.nomCl = dataRow[1].ToString();

this.societe = dataRow[0].ToString();

this.prenomCl = dataRow[2].ToString();

this.adresseCl = dataRow[3].ToString();

this.villeCl = dataRow[4].ToString();

this.codePostCl = dataRow[5].ToString();

return this;

}

else

return null;

}

catch (MonException e)

{

throw new MonException("Erreur sur lecture du client.", "Utilisateur.lectureClient()", e.Message);

}

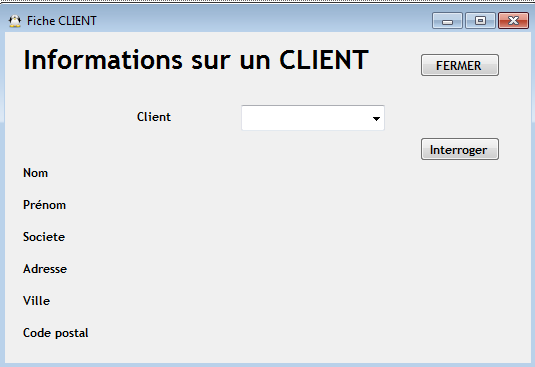
}

}

}

#### Création de la fiche Clientel

* Créez une nouvelle fiche nommée FCLIENT



Dessin de l’écran associé au traitement

- Créez les objets de la fiche et ajoutez trois étiquettes (Tlabel) en face de Société du Client, de nom du client et de Ville du client que vous nommerez respectivement LB\_SOCIETE, LB\_NOM, et LB\_VILLE. Leur champ text sera initialisé à vide.

- Le chargement de la boîte déroulante se fera dans la partie **constructeur** de la classe Fclient

- Le déclenchement du traitement se fera par l’intermédiaire du bouton

Spécifications du traitement

Cette fiche va communiquer avec la couche métier, elle lui passera et récupèrera les données.

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using Utilitaires;

using Metier;

namespace Commercial.Presentation

{

public partial class FClient : Form

{

public FClient()

{

InitializeComponent();

List<String> mesNumeros;

try

{

// Lecture classique

cb\_Client.DataSource = unClient.LectureNoClient();

Clientel unClient = new Clientel();

mesNumeros = unClient.LectureNoClient();

foreach (String item in mesNumeros)

{

cb\_Client.Items.Add(item);

}

// Lecture à partir d’un data source

cb\_Client.DataSource = unClient.LectureNoClient();

}

catch (MonException exception)

{

MessageBox.Show(exception.MessageApplication(), exception.Message);

}

}

private void cb\_Client\_SelectedIndexChanged(object sender, EventArgs e)

{

}

private void bt\_Fermer\_Click(object sender, EventArgs e)

{

Close();

}

private void bt\_Interroger\_Click(object sender, EventArgs e)

{

string numCli;

Clientel unClient = new Clientel();

Clientel unClientCherche;

try

{

numCli = cb\_Client.Text;

unClientCherche = unClient.RechercheUnClient(numCli);

lb\_Nom.Text = "Nom : " + unClientCherche.NomCl;

lb\_Prenom.Text = "Prénom : " + unClientCherche.PrenomCl;

lb\_Societe.Text = "Societe : " + unClientCherche.Societe;

lb\_Adresse.Text = "Adresse : " + unClientCherche.AdresseCl;

lb\_Ville.Text = "Ville : " + unClientCherche.VilleCl;

lb\_Cp.Text = "Code Postal :" + unClientCherche.CodePostCl;

}

catch (MonException exception)

{

MessageBox.Show(exception.MessageApplication(), exception.Message);

}

}

}

}

Bouton Fermer

Sur la fermeture de la fiche, il faut revenir à la fiche principale.

private void bt\_close\_Click(object sender, EventArgs e)

{

Close();

}

#### Appel de la fiche Client dans le menu de la fiche principale

private void Menu\_fiche\_client\_Click(object sender, EventArgs e)

{

FClient fc;

fc = new FClient();

fc.ShowDialog();

}

#### Travail à faire

#### Ajoutez ce traitement à votre application. Il sera déclenché par un item du menu.