C# - FICHE PRATIQUE N° 3

Une petite gestion de personnel
Les bases de l'accès aux données.

A/ Création de la base de données

1. Base de données Gesper

Créez une base de données SqlServer dont les tables seront préfixées par « tp1 » :

Voici le script de création de la base :

```
create table tp1_service (
    code char(2) not null,
    designation varchar(30),
    primary key (code)
);
create table tp1_employe (
    numero char(36) not null default newid(),
    nom varchar(30),
    prenom varchar(25),
    sexe char(1),
    cadre bit default 0,
    salaire money,
    sce char(2),
    primary key (numero),
    foreign key (sce) references tp1_service (code)
);
```

B/ Première approche

Lancez VS et créez un nouveau projet nommé "Fiche3".

Renommez le formulaire principal : "Fm_service", et enregistrez tout.

1. L'accès à la base de données

Il existe plusieurs méthodes pour accéder à une base de données. Nous allons utiliser la plus courante : le mode déconnecté. Nous reverrons cela par la suite, mais le principe est le suivant :

- Un objet « connexion » permet de se connecter à une base de données.
- Un objet « adaptateur », lié à la connexion, permet d'indiquer quelles données doivent être accédées.
- Un objet « dataSet » (groupe de données) est rempli à partir de l'adaptateur.
- Le dataSet obtient ses données depuis le SGDB, et les mémorise localement, la connexion est alors coupée.
- On travaille sur ces données locales (consultation bien sûr, mais également mise à jour).
- La validation de l'ensemble des mises à jours se fait en une fois, à l'aide d'une méthode de l'adaptateur.

> Déposez un « SqlDataAdapter » sur le formulaire, l'assistant se lance :

- Créez une nouvelle connexion
- Sélectionnez votre serveur SQLServer (ne rien sélectionner s'il s'agit d'un serveur local)

- Réglez les paramètres de sécurité de connexion (en pratique souvent « authentification Windows »)
- Sélectionnez votre base de données et testez la connexion
- Continuez le paramétrage de l'adaptateur en utilisant le générateur de requêtes pour obtenir « select * from tp1_service ».
- Cliquez sur le bouton « Options avancées » et décochez « Actualiser le DataSet ».
- Renommez les éléments générés : dbCo_gesper pour la connexion, dbAd_service pour l'adaptateur.

> Observez les propriétés de ces deux objets et réfléchissez, consultez l'aide et posez des questions ! Les instructions insert, delete et update seront utilisées pour la mise à jour des données, il est parfois nécessaire de les modifier…

> Faites un clic droit n'importe où dans la fenêtre de conception et choisissez générer le groupe de données dans le menu données (il s'agit en fait du dataSet). Nommez-le dbDs_service. En fait, il y a création d'un type de données dbDs_service et d'un objet de ce type, dbDs_service1 qui est ajouté au formulaire. Les groupes de données peuvent être visualisés par l'intermédiaire de l'explorateur de solution.

Vous êtes prêt à accéder aux données de la base SQL Server.

2. Les contrôles liés aux données

L'accès aux données peut être réalisé par l'intermédiaire des contrôles Windows traditionnels. Chacun de ces contrôles est en fait "connecté" à un des champs gérés par un composant dataSet.

Ajoutez quelques services dans votre base de données en utilisant l'analyseur de requêtes fourni avec SQL Server, puis continuez votre formulaire :

> Ajoutez deux Labels et deux TextBox (tb_code et tb_designation) pour afficher le code et la désignation de chaque service.

> La liaison des TextBox avec le dataSet se fait à l'aide de leur propriété databindings/Text.

> Il reste à remplir le dataSet avec les données de la table service, vous pouvez le faire dans le constructeur du formulaire :

```
public Fm_service()
{
    //
    // Requis pour la prise en charge du Concepteur Windows Forms
    //
    InitializeComponent();
    //
    // TODO : ajoutez le code du constructeur après l'appel à InitializeComponent
    //
    db_Ds_service1.Clear();
    dbAd_service.Fill(db_Ds_service1);
}
```

> Exécutez l'application, le premier service est affiché !

3. Améliorations diverses

Nous allons progressivement obtenir le formulaire ci-dessous :

> Ajoutez des boutons de navigation

La navigation est assurée par les boutons bt_premier, bt_suivant, bt_precedent et bt_dernier. Leur code est basé sur la propriété bindingContext du formulaire (contexte de liaison de données). A titre d'exemple, voici celui du premier bouton :

```
private void bt_premier_Click(object sender, System.EventArgs e)
{
    this.BindingContext[dbDs_service1,"tp1_service"].Position=0;
    // ou encore : this.BindingContext[dbDs_service1, dbDs_service1.tp1_service.TableName].Position=0;
}
```

La propriété this.BindingContext[dbDs_service1,"tp1_service"].Count contient le nombre d'enregistrements.

> Ajoutez un indicateur de position et un compteur d'enregistrements

Il faut écrire une méthode affichePosCpt qui permet d'afficher la position courante et le nombre d'enregistrements dans un TextBox (tb_posCpt).

Cette méthode doit être appelée partout où la position et/ou le nombre d'enregistrements peut varier.

> Ajout et suppression

La propriété this.BindingContext[dbDs_service1,"tp1_service"] est du type CurrencyManager (ou PropertyManager), classe dérivée de BindingManagerBase. Regardez l'aide à propos de la classe CurrencyManager…

> Validation des modifications (première approche)

Pour le moment, toutes les modifications, ajouts ou suppressions se font dans le dataSet local à l'application, la base de données n'est donc pas affectée. La méthode update de l'adaptateur permet de les appliquer en une seule fois, à l'aide de ses commandes insert, update et delete. Ces commandes sont générées par VS mais il est bien entendu possible de les modifier en cas de besoin.

Avant d'effectuer la mise à jour de la base de données, il est nécessaire de valider les modifications effectuées par l'intermédiaire de l'interface dans le dataSet local. Ceci peut être fait de deux manières :

- Ces modifications sont validées automatiquement lorsque l'on change de position dans le dataSet.
- Il est possible d'appeler la méthode « EndCurrentEdit » pour les valider explicitement.

Remarque : la méthode « CancelCurrentEdit » permet au contraire de les annuler.

Voici le code du bouton de validation (bt_ok) :

```
private void bt_ok_Click(object sender, System.EventArgs e)
{
    this.BindingContext[dbDs_service1,"tp1_service"].EndCurrentEdit();
    dbAd_service.Update(dbDs_service1,"tp1_service");
}
```

- Seul un clic sur ce bouton met à jour la base de données.
- Si l'utilisateur quitte l'application sans cliquer sur ce bouton, les modifications sont perdues.

Le fonctionnement global est donc le suivant :

- L'utilisateur ajoute, supprime et modifie les données d'une (ou de plusieurs) table(s) par l'intermédiaire d'un (ou de plusieurs) formulaire(s).
- Il valide l'ensemble des modifications en une fois, en cliquant sur le bouton de validation. Si cette validation provoque des erreurs (non respect d'une contrainte), il faut les traiter une par une en demandant à l'utilisateur de modifier ses entrées.

> Autre solution pour la validation

Une autre solution consiste à mettre à jour automatiquement la base de données à chaque mise à jour du DataSet. Il suffit pour cela de programmer la réaction des DataTables du DataSet aux événements « RowChanged » pour l'ajout et la modification, et « RowDeleted » pour la suppression. Voici le code à ajouter pour la table service :

- Méthodes de gestion des événements « RowChanged » et « RowDeleted » :

```
private void tp1_service_RowChanged(object sender, DataRowChangeEventArgs e)
{
    if((e.Action==DataRowAction.Add)||(e.Action==DataRowAction.Change))
    {
        dbAd_service.Update(new DataRow[] {e.Row});
    }
}

private void tp1_service_RowDeleted(object sender,DataRowChangeEventArgs e)
{
    if (e.Action==DataRowAction.Delete)
    {
        dbAd_service.Update(new DataRow[] {e.Row});
    }
}
```

- Mise en place de la gestion de ces événements dans le constructeur du formulaire :

```
public Fm_service()
{
    ...
    dbAd_service.Fill(dbDs_service1);
    dbDs_service1.tp1_service.RowChanged += new DataRowChangeEventHandler(tp1_service_RowChanged);
    dbDs_service1.tp1_service.RowDeleted += new DataRowChangeEventHandler(tp1_service_RowDeleted);
}
```

- Le bouton OK ne doit plus mettre à jour la base de données :

```
private void bt_ok_Click(object sender, System.EventArgs e)
{
    this.BindingContext[dbDs_service1,"tp1_service"].EndCurrentEdit();
}
```

Cette seconde solution a l'avantage de mettre à jour de manière certaine et automatique la base de données. De plus, les éventuelles erreurs sont détectées immédiatement et peuvent être traitées de manière plus simple. Le seul inconvénient de cette méthode est que le SGBD est plus sollicité. Les avantages semblent l'emporter, cette méthode sera utilisée systématiquement dans cette série d'exercices.

> Gestion des problèmes de validation

Deux types de problèmes peuvent se présenter :

- Lors de la modification d'une table du DataSet local, une contrainte locale peut ne pas être respectée (unicité de clé, contrainte de clé étrangère, …). Il faut donc gérer le problème à chaque mise à jour des données locales, c'est-à-dire lorsque l'on change de position dans le DataSet boutons, premier, suivant,…) ou que l'on fait appel à la méthode « EndCurrentEdit ». Voici l'exemple du bouton OK :

```
private void bt_ok_Click(object sender, System.EventArgs e)
{
    try
    {
        this.BindingContext[dbDs_service1,"tp1_service"].EndCurrentEdit();
    }
    catch (System.Exception pb)
    {
        MessageBox.Show(pb.Message);
    }
}
```

- Lors de la validation de la modification à destination du SGBD. Là encore, une contrainte peut ne pas être respectée. Le même principe de précaution doit donc être appliqué lors du traitement des événements « RowChanged » et « RowDeleted » des tables du DataSet. Exemple de l'événement « RowDeleted » :

```
private void tp1_service_RowDeleted(object sender, DataRowChangeEventArgs e)
{
    if (e.Action==DataRowAction.Delete)
    {
        try
        {
            dbAd_service.Update(new DataRow[] {e.Row});
        }
        catch (Exception pb)
        {
            MessageBox.Show(pb.ToString());
            e.Row.RejectChanges();
        }
    }
}
```

La méthode « RejectChanges » de la classe « DataRow » permet d'annuler la modification réalisée dans la table locale.

C/ Le composant DbNavigateur

1. Présentation

Il s'agit d'un composant prêt à l'emploi qui implémente l'ensemble des techniques de navigation étudiées. Il vous évitera de réécrire systématiquement les mêmes méthodes.

Il fournit également une méthode statique « GererRowAction » qui simplifie l'écriture des gestionnaires d'événement « RowChanged » et « RowDeleted ».

La gestion des erreurs de validation locales ou vers le SGBD est prise en charge par ce composant.

2. Mode d'emploi

Pour d'installer le composant dans la boîte à outils, il faut utiliser l'option « ajouter des éléments dans la boîte à outils » du menu « outils »,.

> Initialisation

Une fois le composant intégré sur un formulaire (nommé ici dbNv_service), il suffit de l'initialiser sur l'événement load :

```
dbNv_service.Init(   this.BindingContext[dbDs_service1,"tp1_service"],   // Currency Manager
                     new Navigation.OnDbEventFunction(surAjout),
                     new Navigation.OnDbEventFunction(surDeplact));
```

Les deux derniers paramètres sont optionnels, surAjout est le nom d'une méthode que vous voulez voir exécutée à chaque ajout d'un enregistrement (donner le focus à un champ par exemple…), surDeplact est le nom d'une méthode que vous voulez voir exécutée lorsque l'on passe d'un enregistrement à un autre.

Remarque : pour programmer une méthode « surDeplact » sans programmer de méthode « surAjout », il faut écrire :

```
dbNv_service.Init(   this.BindingContext[dbDs_service1,"tp1_service"],   // Currency Manager
                     null,
                     new Navigation.OnDbEventFunction(surDeplact));
```

> Traitement des événements « RowChanged » et « RowDeleted » :

Il suffit d'écrire les deux gestionnaires ci-dessous pour chacune des tables du DataSet.

```
private void tp1_service_RowChanged(object sender, DataRowChangeEventArgs e)
{
    DbNavigateur.GererRowAction(e,DbAd_service);
}

private void tp1_service_RowDeleted(object sender, DataRowChangeEventArgs e)
{
    DbNavigateur.GererRowAction(e,DbAd_service);
}
```

Essayez ce composant sur votre formulaire…