C# - FICHE PRATIQUE N° 5

Une petite gestion de personnel
Accès aux données centralisé, utilisation d'une relation.

A/ Un dataset commun

1. Le problème

Nous avons vu que la solution retenue à la fiche 4 consiste à maintenir deux copies de la table service dans l'application.

Outre la place perdue, cette situation provoque des problèmes de cohérence. Il serait beaucoup plus logique de n'utiliser qu'un seul dataset commun à tous les formulaires de l'application.

2. La solution

Si le dataset est unique, il doit logiquement être créé sur le formulaire principal, et être alimenté par deux dataAdapter : un pour la table service et un autre pour la table employé. (En fait, ces objets pourraient être créés sur un (ou plusieurs) autre(s) formulaire(s) spécialisé(s) dans l'accès aux données).

Créez sur le formulaire principal les deux dataAdapter (dbAd_service et dbAd_employe), le dataset dbDs_empSce (alimenté par les deux dataAdapter), le tout en réutilisant la connexion existante (dbCo_gesper). ATTENTION : ne demandez pas à actualiser le DataSet (bouton « options avancées »).

Supprimez les objets d'accès aux données créés sur les autres formulaires à l'exception du sqlCommand dbPs_idEmploye destiné à appeler la procédure stockée SqlServer.

Faites passer le slqCommand dbPs_idEmploye du formulaire Fm_employe au formulaire principal (couper/coller). Il est maintenant possible de paramétrer graphiquement sa connexion.

Lorsque les objets dataset ont été générés, VS a créé automatiquement une classe leur correspondant. Ces classes peuvent être visualisées dans les fichiers dbDs_employe.cs et dbDs_service.cs. Utilisez l'explorateur de solution pour supprimer ces définitions.

3. Aménagement du code

- Les instructions de remplissage des datasets dans les constructeurs de Fm_service et Fm_employe doivent être supprimées, le remplissage doit maintenant être effectué dans le constructeur du formulaire principal.

- Il en va de même pour le code permettant la gestion des événements « RowChanged » et « RowDeleted » des deux tables du DataSet.

- Le bouton recharger devrait être inutile dans notre nouvelle version. Supprimez-le ainsi que la procédure événementielle associée au clic sur ce bouton.

- L'initialisation du dbNavigateur de chaque formulaire nécessite un DataSet. Elle a actuellement lieu dans la méthode « load » du formulaire. En fait, il est plus simple de la réaliser dans le constructeur de la classe, il faut donc le surcharger en créant un constructeur paramétré ayant accès au DataSet du formulaire principal.

La solution suivante est assez générale (exemple pour la classe Fm_service à transposer pour la classe Fm_employe) :

. Supprimez la méthode « load » ainsi que sa référence (événement du formulaire).

. Déclarez une donnée permettant de mémoriser le DataSet fourni par le formulaire principal :

```
dbDs_empSce dbDs;
```

. Créez un nouveau constructeur pour la classe Fm_service :

```
public Fm_service(dbDs_empSce p_dbDs):this()
{
    dbDs=p_dbDs;
    dbNv_service.Init(BindingContext[dbDs,"tp1_service"]);
}
```

Ce constructeur appelle le constructeur par défaut, renseigne la propriété « dbDs » et initialise le dbNavigateur. (Le constructeur par défaut est appelé à l'aide du mot-clé « this »).

. Il reste à appeler ce nouveau constructeur dans le constructeur de la classe Fm_principal lors de l'instanciation de l'objet fsce :

```
public Fm_principal()
{
    InitializeComponent();
    // remplissage du dataset
    dbDs_empSce1.Clear();
    dbAd_service.Fill(dbDs_empSce1,"tp1_service");
    dbAd_employe.Fill(dbDs_empSce1,"tp1_employe");
    // Construction du formulaire service
    fsce=new Fm_service(dbDs_empSce1);
}
```

Remarque : on ne peut pas faire appel à ce constructeur lors de la déclaration du formulaire « fsce » puisque l'objet « dbDs_empSce1 » n'est créé qu'à l'instanciation du formulaire principal. Il faut donc obligatoirement l'instancier dans le constructeur du formulaire principal.

4. Rétablissement des liaisons de données

Bien évidemment, les contrôles des deux formulaires ne sont plus liés aux données. Cette liaison ne peut plus être faite graphiquement et doit être rétablie par programme, dans le constructeur de chaque formulaire.

. Exemple du formulaire Fm_service :

```
public Fm_service(dbDs_empSce p_dbDs):this()
{
    dbDs=p_dbDs;
    dbNv_service.Init(BindingContext[dbDs,"tp1_service"]);
    tb_code.DataBindings.Add("Text",dbDs,"tp1_service.code");
    tb_designation.DataBindings.Add("Text",dbDs,"tp1_service.designation");
}
```

Remarque : une liaison de données peut être dynamiquement modifiée comme dans l'exemple ci-dessous.

```
this.tb_designation.DataBindings.Remove(tb_designation.DataBindings["Text"]);
this.tb_designation.DataBindings.Add("Text",dbDs,"tp1_service.code");
```

. Formulaire Fm_employe :

Les boutons d'options n'ont pas à être liés puisqu'ils sont mis à jour par la méthode « SurDeplact ».

La liaison de la liste des services est particulière : le type de liaison est « SelectedValue » et non « Text ». Pour la case à cocher, il s'agit de « Checked ».

Pour le chargement de la liste des services, les propriétés « DisplayMember » et « ValueMember » peuvent encore être renseignée par l'intermédiaire du concepteur graphique, par contre, la propriété « DataSource » doit être renseignée par programme, toujours dans le constructeur. On obtient finalement le constructeur suivant :

```
public Fm_employe(dbDs_empSce p_dbDs):this()
{
// mémorisation du dataSet permettant l'accès aux données
    dbDs=p_dbDs;
// initialisation du dbNavigateur
    dbNv_employe.Init(     BindingContext[dbDs,"tp1_employe"],
                           new Navigation.OnDbEventFunction(surAjout),
                           new Navigation.OnDbEventFunction(surDeplact));
// mise en place des liaisons de données
    this.tb_numero.DataBindings.Add("Text", dbDs,"tp1_employe.numero");
    this.tb_nom.DataBindings.Add("Text", dbDs,"tp1_employe.nom");
    this.tb_prenom.DataBindings.Add("Text", dbDs,"tp1_employe.prenom");
    this.tb_salaire.DataBindings.Add("Text", dbDs,"tp1_employe.salaire");
    this.cb_cadre.DataBindings.Add("Checked", dbDs,"tp1_employe.cadre"); // Checked, pas Text
// cas particulier de la liste des services
    this.cb_service.DataSource= dbDs.tp1_service;
    this.cb_service.DataBindings.Add("SelectedValue", dbDs,"tp1_employe.sce");
}
```

5. Test et mise au point

Voilà, cela fonctionne correctement ! Il reste une toute petite chose, si on ajoute un service sans valider avant de quitter le formulaire, celui-ci n'est pas pris en compte. Pour mettre en place une « validation automatique » lorsque l'utilisateur referme le formulaire, il suffit d'ajouter au code du bouton retour, un appel à la méthode « Valider » du dbNavigateur (la même chose peut être faite pour le formulaire Fm_employe).

Autre remarque : dans certains cas, la liste déroulante liée au service de l'employé présente un défaut d'affichage lorsque le champ « sce » a la valeur nulle (affichage du premier service de la liste). Pour régler le problème, il suffit de modifier très légèrement la procédure « surDeplact » :

```
private void surDeplact()
{
    if (dbNv_employe.Position!=-1)   // la position est-elle valide ?
    {
        if (dbNv_employe.Courant()["sexe"].ToString()=="f")
            rb_feminin.Checked=true;
        else
            rb_masculin.Checked=true;
        if (dbNv_employe.Courant()["sce"]==DBNull.Value)
            cb_service.SelectedIndex=-1;
    }
}
```

B/ Un troisième formulaire

Il s'agit maintenant de créer un formulaire permettant de manipuler en même temps les services et les employés. Il faut bien entendu que les informations soient liées en fonction du code service de chaque employé.

1. Création d'une relation

L'un des intérêts du dataSet, c'est qu'il prend en charge les relations existant entre les tables.

- Affichez le schéma du dataSet dbDs_empsce1.
- Sélectionnez l'une des deux tables et choisissez « Ajouter … une relation » dans le menu contextuel.
- Paramétrez la relation comme indiqué ci-dessous :

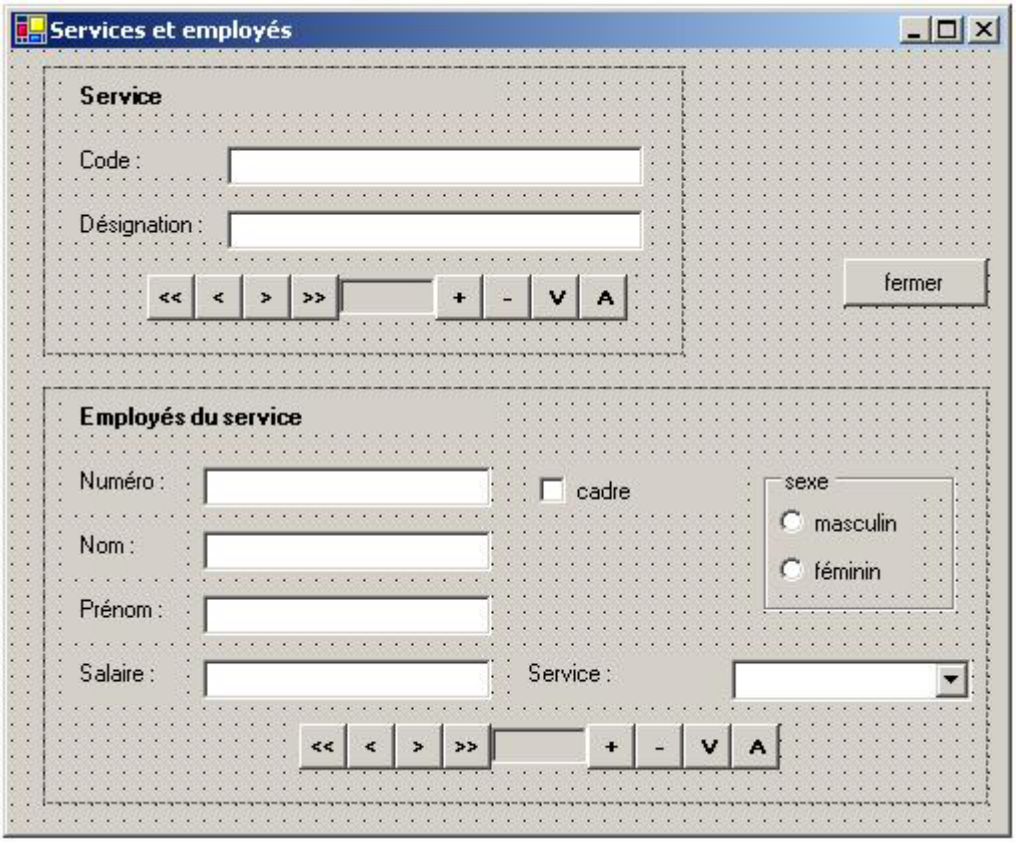
. Nom : serviceemploye
. Elément parent : tp1_service
. Elément enfant : tp1_employe
. Champ clé : code
. Champ clé étrangère : sce
. Règle de mise à jour : none
. Règle de suppression : none
. Règle d'acceptation/de rejet : cascade

Les règles de suppression et de mise à jour permettent de paramétrer le comportement de la relation lors de la suppression et de la modification de la clé d'un enregistrement de la table parent.

- Refermez le schéma en enregistrant les modifications.

2. Que veut-on obtenir ?

Il s'agit de faire un formulaire affichant à la fois les services et les employés.



Le cadre du haut affiche les informations concernant un service, celui du bas affiche les informations concernant les employés affectés au service courant. Un changement de service provoque le rafraîchissement des données concernant les employés.

En fait, comme nous allons le voir, cette synchronisation est prise en charge par la relation créée entre les deux tables.

3. Ajout du formulaire

- Ajoutez un nouveau formulaire « Fm_empSce » à votre application et créez tous les contrôles nécessaires à l'édition des informations des deux tables (vous pouvez également utiliser le copier/coller).

- Mettez en place le code nécessaire à l'ouverture de ce formulaire depuis le formulaire principal.

- Etablissez toutes les liaisons de données, sachant que :

. Le dbNavigateur concernant les employés doit être lié à la relation établie entre les deux tables (celle ci est « vue » comme une donnée de la table parente) :

```
dbNv_employe.Init(    BindingContext[dbDs,"tp1_service.serviceemploye"],
                      new Navigation.OnDbEventFunction(surAjout),
                      new Navigation.OnDbEventFunction(surDeplact));
```

. les contrôles concernant les employés sont également liés à la relation elle-même, exemple pour le nom de l'employé : « this.tb_nom.DataBindings.Add("Text", dbDs,"tp1_service.serviceemploye.nom"); »

- Testez le fonctionnement de l'application. Essayez toutes les manipulations envisageables… Un problème subsiste lorsque l'on ajoute un nouveau service. Ce problème mineur sera bientôt résolu.