

Laboratoire 6

Validation des données
Opérations sur les données
Gestion des conflits d'accès concurrentiels
Enregistrement des données au format XML

Objectifs d'apprentissage


- Valider des données
- Enregistrer et annuler des modifications
- Supprimer et ajouter un enregistrement
- Gérer les conflits d'accès concurrentiels
- Enregistrer les données d'une **DataTable** dans le format **XML**

1. Valider des données simples (données dans un champ de saisie)

En programmation, la validation de données est une étape importante. Dans ce laboratoire, nous allons valider des données tapées par l'utilisateur dans une zone de texte.

- Copiez votre labo5 et renommez le *labo6*:
- Sur le formulaire **frmListeDeValeurs**, modifiez les contrôles suivants :

Ancien nom	Nouveau nom	Propriété
conNoTextBox	txtNoContrat	<i>ReadOnly : True</i>
conDateTimePicker	dtpDateContrat	<i>Enabled : False</i>
conAcompteTextBox	txtAcompte	
conMontantTextBox	txtMontant	
conPayeTextBox	txtMontantPaye	
conTypeOccTextBox	txtNbPlaces	<i>ReadOnly : True</i>

- Ajoutez, sur le formulaire, un contrôle *ErrorProvider*. Nous utiliserons ce contrôle pour afficher nos messages d'erreur. Nommez-le **errMessage**.  **ErrorProvider**
- Avant de programmer la validation ajoutez les espaces de noms suivants :
using System.Text.RegularExpressions;
using System.Globalization;

- programmez l'événement *Validating* de la zone de texte **txtMontantPaye** comme suit :

```
public partial class frmListeDeValeurs : Form
{
    public Regex exprMontant { get; } // déclare une variable en lecture seulement

    public frmListeDeValeurs()
    {
        InitializeComponent();

        char point = Convert.ToChar(CultureInfo.CurrentCulture.NumberFormat.NumberDecimalSeparator);
        exprMontant = new Regex("^([0-9]{1,4})" + point + "[0-9]{2})?$");
    }

    private void txtMontantPaye_Validating(object sender, CancelEventArgs e)
    {
        string messageErreur = string.Empty;
        if (!exprMontant.IsMatch(txtMontantPaye.Text))
        {
            messageErreur = "Cette donnée n'est pas dans un format valide";
            e.Cancel = true;
        }
        else
        {
            double montantPaye = Convert.ToDouble(txtMontantPaye.Text);
            double montant = Convert.ToDouble(txtMontant.Text);
            double acompte = Convert.ToDouble(txtAcompte.Text);
            if (montantPaye > montant)
            {
                messageErreur = "Le montant payé dépasse le montant du contrat";
                e.Cancel = true;
            }
            else if (montantPaye < acompte)
            {
                messageErreur = "Le montant payé est inférieur au montant de l'acompte";
                e.Cancel = true;
            }
        }

        errMsg.SetError(txtMontantPaye, messageErreur);
    }
}
```

Notez que l'événement *Validating* a lieu à chaque fois qu'un contrôle est sur le point de perdre le focus et que l'instruction, *e.cancel = true* annule la perte du focus. Ceci force l'utilisateur à entrer une donnée valide.

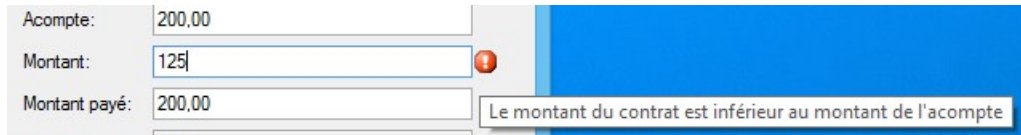
- Exécutez. Tapez un montant non valide dans la zone de texte du montant payé puis cliquez ailleurs. Observez le ! sur fond rouge qui clignote. Survolez ce ! et observez qu'il y a un message d'erreur. Observez également que la zone de texte ne perd pas le focus lorsqu'il y a un message d'erreur.



- Testez en entrant une donnée valide.

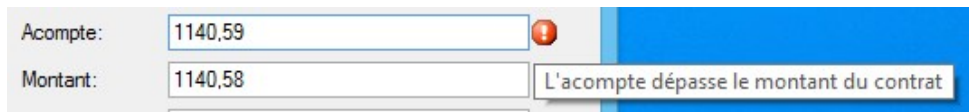
- Programmez l'événement *Validating* de la zone de texte **txtMontant**
Le format du montant du contrat doit être le même que celui du montant payé. De plus, ce montant doit être supérieur ou égal au montant de l'acompte et supérieur ou égal au montant payé. (revoir supérieur)

Si ce n'est pas valide, vous devez afficher un message d'erreur adéquat et annuler la perte de focus.



- Programmez l'événement *Validating* de la zone de texte **txtAcompte**
Le format de l'acompte doit être le même que celui du montant payé. De plus, ce montant doit être inférieur ou égal au montant du contrat et inférieur ou égal au montant payé.

Si ce n'est pas valide, vous devez afficher un message d'erreur adéquat et annuler la perte de focus.



- Notez que l'événement *Validated* a lieu quand la validation est réussie
- Sur le formulaire, ajoutez un bouton **btnFermer** puis programmez son événement *Click* comme suit :

```
private void btnFermer_Click(object sender, EventArgs e)
{
    this.Close();
}
```
- Exécutez l'application, tapez une donnée invalide dans n'importe quelle zone de texte puis cliquez sur le bouton **btnFermer**. Observez que vous ne pouvez pas le faire.
- Maintenant, affectez la valeur *False* à la propriété *CausesValidation* du bouton **btnFermer**.
- Exécutez l'application, tapez une donnée invalide dans n'importe quelle zone de texte puis cliquez sur le bouton **btnFermer**. Observez que la fenêtre se ferme.

La propriété **CausesValidation** d'un contrôle est très importante. Elle annule l'appel à l'événement *Validating* d'un autre contrôle.

2. Valider des données dans un *DataGridView*

Pour valider une donnée dans un *DataGridView*, il existe quatre (4) événements importants :

- **DataError** : Cet événement est appelé à chaque fois qu'une cellule contient une valeur qui ne correspond pas au type de la donnée attendue (dans le *DataGridView*, les données sont fortement typées).
- **Validating** : Cet événement est appelé à chaque fois que le *DataGridView* perd le focus. On l'utilise pour valider, en même temps, toutes les données du *DataGridView*.
- **RowValidating** : Cet événement est appelé à chaque fois qu'une ligne perd le focus à l'intérieur du *DataGridView*. On l'utilise pour valider une ligne à la fois.
- **CellValidating** : Cet événement est appelé à chaque fois qu'une cellule perd le focus à l'intérieur du *DataGridView*. On l'utilise pour valider une donnée à la fois.

Ici, nous allons programmer deux événements **DataError** et **RowValidating**.

L'événement *DataError* est toujours appelé avant l'événement *RowValidating*.

- Sur le formulaire **frmListeDeValeursTab**, modifiez les contrôles suivants (nous allons utiliser ces noms de colonnes plus tard).

Ancien nom	Nouveau nom	Propriété
dataGridViewTextBoxColumn1	dgTxtNoContrat	<i>ReadOnly : True</i>
dataGridViewTextBoxColumn2	dgTxtDateContrat	<i>ReadOnly : True</i>
dataGridViewTextBoxColumn3	dgTxtAcompte	
dataGridViewTextBoxColumn4	dgTxtMontant	
dataGridViewTextBoxColumn5	dgTxtMontantPaye	
dataGridViewTextBoxColumn6	dgTxtNbClients	<i>ReadOnly : True</i>

- Programmez l'événement *DataError* du *DataGridView* **dgContrats** comme suit :

```
private void dgContrats_DataError(object sender, DataGridViewDataErrorEventArgs e)
{
    dgContrats.Rows[e.RowIndex].ErrorText =
        "Le type de données de " + dgContrats.Columns[e.ColumnIndex].HeaderText + " n'est pas valide.";
    e.Cancel = true;
}
```

Cette instruction affiche un message d'erreur dans un contrôle de type *ErrorProvider* au début de la ligne qui contient l'erreur. Cette validation est automatique.

- Exécutez l'application puis, dans une cellule, tapez une donnée dont le type n'est pas valide puis sélectionnez une autre cellule. Observez le message d'erreur au début de la ligne.
- Programmez l'événement **RowValidating** du *DataGridView* dgContrats comme

```
S
private void dgContrats_RowValidating(object sender, DataGridViewCellCancelEventArgs e)
{
    string messageErreur = string.Empty;
    string strMontant = dgContrats["dgTxtMontant", e.RowIndex].Value.ToString();
    string strMontantPaye = dgContrats["dgTxtMontantPaye", e.RowIndex].Value.ToString();
    string strAcompte = dgContrats["dgTxtAcompte", e.RowIndex].Value.ToString();

    if (string.IsNullOrEmpty(strMontant) || string.IsNullOrEmpty(strMontantPaye) ||
        string.IsNullOrEmpty(strAcompte))
    {
        messageErreur = "Aucun montant ne doit être vide";
    }
    else
    {
        double montantPaye = Convert.ToDouble(strMontantPaye);
        double montant = Convert.ToDouble(strMontant);
        double acompte = Convert.ToDouble(strAcompte);
        if (new List<double> {montantPaye, acompte, montant}.Any(d => d < 0))
            // équivalent à if((montantPaye<0)|| (acompte<0)|| (montant<0))
        {
            messageErreur = "Aucun montant ne doit être plus petit que 0";
        }
        else if (acompte > montant)
        {
            messageErreur = "L'acompte ne doit pas dépasser le montant du contrat";
        }
        else if (acompte > montantPaye)
        {
            messageErreur = "L'acompte ne doit pas dépasser le montant payé";
        }
        else if (montant < montantPaye)
        {
            messageErreur = "Le montant du contrat ne doit pas être inférieur au montant payé";
        }
    }

    dgContrats.Rows[e.RowIndex].ErrorText = messageErreur;
    if (messageErreur != string.Empty)
    {
        e.Cancel = true;
    }
}

{
    messageErreur = "L'acompte ne doit pas dépasser le montant payé";
    e.Cancel = true;
}
else if (montant < montantPaye)
{
    messageErreur = "Le montant du contrat ne doit pas être inférieur au montant payé";
    e.Cancel = true;
}
}

dgContrats.Rows[e.RowIndex].ErrorText = messageErreur;
}
```

e.RowIndex et *e.ColumnIndex* contiennent respectivement le numéro de ligne et de colonne sur laquelle il y a eu une erreur.

- Exécutez l'application, tapez une donnée invalide quelconque sur une des lignes du *DataGridView* puis sélectionnez une autre ligne. Observez le message d'erreur au début de la ligne.

3. Enregistrer / annuler des modifications

Les modifications que vous effectuez sont enregistrées dans la *DataTable* **contrat**. Pour enregistrer les données dans la vraie base de données, vous devez cliquer sur le bouton *Enregistrer les données* dans la barre de navigation.



Voici le code associé à ce bouton :

```
private void contratBindingNavigatorSaveItem_Click(object sender, EventArgs e)
{
    this.Validate();
    this.contratBindingSource.EndEdit();
    this.tableAdapterManager.UpdateAll(this.bDVoyagesDataSet);
}
```

this.validate() : assure la validation du formulaire courant en appelant les méthodes *validating* et *validated* de tous les contrôles du formulaire.

Si les données sont valides, le contrôleur de la *DataTable* **contrat** met fin à l'édition, c'est-à-dire qu'il sauvegarde dans la *DataTable* **contrat**, la donnée que l'utilisateur est en train d'éditer dans la vue.

Finalement, le *tableAdapterManager* (cet objet gère tous les *TableAdapters* qui ont été créés) met à jour, dans la vraie base de données, toutes les données qui ont été modifiées dans le *DataSet*.

Ajoutez un bouton **btnEnregistrer** en bas du formulaire **frmListeValeurs**.



- Programmez l'événement *click* de ce bouton comme suit :

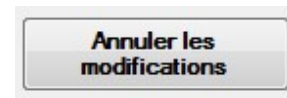
```
private void btnEnregistrer_Click(object sender, EventArgs e)
{
    this.Validate();
    this.contratBindingSource.EndEdit();
    this.contratTableAdapter.Update(this.bDVoyagesDataSet.contrat);

    MessageBox.Show("Les modifications ont été enregistrées dans la base de données.", "Enregistrement des données", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```


- Exécutez l'application, faites des modifications quelconques puis cliquez sur le bouton **btnEnregistrer**.
- Pour vérifier si les modifications ont bien été enregistrées dans la vraie table **contrat**, mettez fin à l'exécution de l'application puis exécutez-la de nouveau.

Lors de l'enregistrement des modifications dans la base de données, Le *TableAdapter* dispose d'un indicateur d'état de chaque ligne (ajoutée, supprimée, modifiée) du *DataTable* correspondant. Les données initiales étant aussi conservées, il nous est toujours possible d'annuler les modifications en cours.

- Ajoutez un bouton de commande **btnAnnuler**.



- Programmez l'événement *click* de ce bouton de la manière suivante.

```
private void btnAnnuler_Click(object sender, EventArgs e)
{
    bDVoyagesDataSet.contrat.RejectChanges();
    // Pour rafraichir les vues
    contratBindingSource.ResetBindings(false);
    MessageBox.Show("Les modifications depuis le dernier enregistrement ont été annulées.", "Annulation des modifications", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

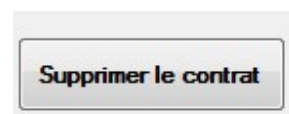
- Exécutez l'application, faites des modifications quelconques incluant la suppression de contrats puis cliquez sur le bouton **btnAnnuler**. Observez que toutes les modifications que vous avez effectuées depuis la dernière sauvegarde sont annulées.
- Nous ne voulons pas que l'utilisateur puisse ajouter, supprimer et enregistrer des modifications en utilisant la barre de navigation. Supprimez ces trois boutons.
- Nous allons programmer la suppression ainsi que l'ajout d'un contrat à l'aide de boutons de commandes.



4. Supprimer /ajouter un enregistrement

- Programmons tout d'abord la suppression.

- Ajoutez un bouton **btnSupprimer** en bas du formulaire. Ce bouton va être utilisé pour supprimer, de la *DataTable* **contrat**, le contrat



qui a été sélectionné.

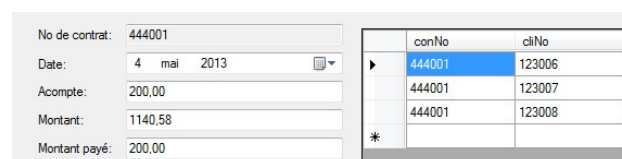
- Programmez l'événement *click* de ce bouton de la manière suivante.

```
private void btnSupprimer_Click(object sender, EventArgs e)
{
    contratBindingSource.RemoveCurrent();
}
```

Ici, nous demandons au contrôleur de supprimer l'enregistrement courant de la *DataTable* **contrat**.

- Exécutez l'application, sélectionnez un contrat quelconque puis cliquez sur le bouton **btnSupprimer**. Observez que ce contrat n'existe plus.
- Maintenant, cliquez sur le bouton **btnEnregistrer** pour enregistrer cette modification. Un message d'erreur s'affiche. Ce message d'erreur provient du fait que, dans la vraie base de données, le numéro de contrat est une clef étrangère dans la table **faitLeVoyage**. Si on supprime ce contrat, l'intégrité référentielle n'est pas respectée (à moins que la suppression en cascade n'ait été prise en compte dans la vraie base de données).
- Pour éviter ce problème, nous allons procéder la manière suivante :
 - Glissez la *DataTable* **faitLeVoyage** (celle qui est située dans la *DataTable* **contrat**) sur le formulaire sous la forme d'un *DataGridView*.

TRÈS IMPORTANT : Vous devez sélectionner la *DataTable* **faitLeVoyage** qui est située dans la *DataTable* **contrat**. Si vous exécutez l'application, dans la *DataTable* **faitLeVoyage**, vous ne devriez voir que les enregistrements qui concernent le contrat sélectionné.



conNo	cliNo
444001	123006
444001	123007
444001	123008

- Affectez la valeur *false* à la propriété *Visible* de ce *DataGridView*.
- Reprogrammez l'événement *click* de ce bouton de la manière suivante et

```
1 référence
private void btnSupprimer_Click(object sender, EventArgs e)
{
    // supprime tous les entegistremments
    // de la DataTable faitLeVoyage et qui sont reliés au contrat
    foreach (var row in faitLeVoyageBindingSource)
    {
        faitLeVoyageBindingSource.Remove(row);
    }
    // supprime le contrat courant de la DataTable contrat
    contratBindingSource.RemoveCurrent();
}
```


- Dans les propriétés du **tableAdapterManager**, assurez-vous que le *TableAdapter* de la table **faitLeVoyage** est présent.
- Complétez la méthode *btnEnregistrer_Click* comme suit :

```
private void btnEnregistrer_Click(object sender, EventArgs e)
{
    this.Validate();
    this.contratBindingSource.EndEdit();

    this.faitLeVoyageBindingSource.EndEdit();
    this.faitLeVoyageTableAdapter.Update(this.bDVoyagesDataSet.faitLeVoyage);

    this.contratTableAdapter.Update(this.bDVoyagesDataSet.contrat);

    MessageBox.Show("les modifications ont été enregistrées dans la base de données.", "Enregistrement");
}
```

Notez bien l'ordre des mises à jour.

- Exécutez l'application, sélectionnez un contrat quelconque puis cliquez sur le bouton **btnSupprimer**. Observez que ce contrat n'existe plus.
- Cliquez sur le bouton **btnEnregistrer**. Le message d'erreur ne s'affiche plus. De plus, tous les enregistrements dans la *DataTable* **faitLeVoyage** qui concernent ce contrat ont été supprimés.

➤ Programmation de **l'ajout**.

- Ajoutez un bouton **btnAjouter** en bas du formulaire.



- Programmez l'événement *click* de ce bouton de la manière suivante.

```
private void btnAjouter_Click(object sender, EventArgs e)
{
    //créer un nouveau contrat en mémoire
    BDVoyagesDataSet.contratRow unContrat = bDVoyagesDataSet.contrat.NewcontratRow();

    // Rechercher le plus gros numéro contrat dans la DataTable contrat
    decimal noContratMax = 0;
    foreach (BDVoyagesDataSet.contratRow uneLigne in bDVoyagesDataSet.contrat.Rows)
        if (uneLigne.conNo > noContratMax) noContratMax = uneLigne.conNo;

    // Valeur du no de contrat = MAX(noContrat)+1
    unContrat.conNo = noContratMax + 1;

    unContrat.conDate = DateTime.Now.Date; // date du contrat = date du jour.
    unContrat.conTypeOcc = 1; // nombre de clients = 1
    unContrat.conMontant = -1; // sera utilisée plus tard
}
```

Notez qu'on pourrait trouver le plus grand numéro de contrat comme suit :

```
decimal noContratMax =  
bDVoyagesDataSet.contrat.Rows.Cast<BDVoyages.contratRow>().Max(r => r.conNo);
```

- Ajouter un nouveau formulaire **frmAjouter** dont le titre est *Ajouter un contrat*. Fixez sa propriété *FormBorderStyle* à *Fixed3D* et sa propriété *MaximizeBox* à *False*
- Ajoutez sur ce formulaire 5 étiquettes sur, 2 zones de texte (**txtMontant** et **txtAcompte**), 3 listes combinées (**cboVoyages**, **cboVendeurs** et **cboClients**) et 2 boutons de commandes (**btnAjouter** et **btnFermer**). Affectez la valeur *DropDownList* à la propriété *DropDownStyle* des 3 listes.
- Liez **cboVoyages** à **noEtDestinationVoyages**, **cboVendeurs** à **noEtNomVendeurs** et **cboClients** à **noEtNomClients**.
- Ajouter l'attribut suivant à la classe **FrmAjouter** :

```
public BDVoyagesDataSet.contratRow unContrat;
```

- Complétez la programmation du bouton **btnAjouter** (Ajouter un Contrat) comme suit :

```
FrmAjouter frmAjout = new FrmAjouter();  
//Passer le contrat au formulaire  
frmAjout.unContrat = unContrat;  
frmAjout.ShowDialog();
```

- Exécutez l'application, puis cliquez sur le bouton **btnAjouter** pour vérifier.

The screenshot displays two windows from a Windows application. The background window, titled 'Liste de valeurs : enregistrement par enregistrement', contains a form with various input fields and dropdown menus. The fields are: 'No de contrat:' (444001), 'Date:' (4 mai 2018), 'Acompte:' (200.00), 'Montant:' (1140.58), 'Montant payé:' (200.00), 'No places:' (3), 'Employé:' (Coté, Claude), 'Voyage:' (3004 (Tadoussac)), and 'Client:' (Sefrest, Wallace). At the bottom of this window are four buttons: 'Ajouter un contrat', 'Supprimer le contrat', 'Annuler les modifications', and 'Enregistrer les modifications'. Overlaid on top of this is a smaller dialog box titled 'Ajouter un contrat'. This dialog box contains four labels with corresponding input fields and buttons: 'Montant:' with a text box and an 'Ajouter' button; 'Acompte:' with a text box and a 'Fermer' button; 'Voyage:' with a dropdown menu showing '3001 (Boston)'; and 'Vendeur:' with a dropdown menu showing 'Bedard, France'. There is also a 'Client:' label with a dropdown menu showing 'Brown, Charlie'.

- Avant de programmer le bouton `btnAjouter` du formulaire `frmAjouter`, faire les étapes suivantes :
 - Ajoutez un contrôle **ErrorProvider** `errMessage`
 - Ajoutez les deux espaces de noms suivants :


```
using System.Text.RegularExpressions;
using System.Globalization;
```
 - Validez les données entrées par l'utilisateur en programmant le clic du bouton `btnAjouter`

B

```
private void btnAjouter_Click(object sender, EventArgs e)
{
    char point = Convert.ToChar(CultureInfo.CurrentCulture.NumberFormat.NumberDecimalSeparator);
    var exprDonnee = new Regex("^([0-9]{1,4})" + point + "([0-9]{2})?");

    if (txtMontant.Text.Trim() == "")
        errMessage.SetError(txtMontant, " le montant ne peut pas être vide");
    else if (!exprDonnee.IsMatch(txtMontant.Text))
        errMessage.SetError(txtMontant, "le montant n'est pas dans un format valide");
    else
    {
        errMessage.SetError(txtMontant, ""); // montant valide
        if (txtAcompte.Text.Trim() == "")
            errMessage.SetError(txtAcompte, "L'acompte ne doit pas être vide");
        else if (!exprDonnee.IsMatch(txtAcompte.Text))
            errMessage.SetError(txtAcompte, "L'acompte n'est pas dans un format valide");
        else
        {
            decimal montant = Convert.ToDecimal(txtMontant.Text);
            decimal acompte = Convert.ToDecimal(txtAcompte.Text);
            if (acompte > montant)
                errMessage.SetError(txtAcompte, "L'acompte ne doit pas dépasser le montant");
            else
            {
                errMessage.SetError(txtAcompte, "");
            }
        }
    }
}
```

- Une fois la validation terminée, affectez les valeurs au contrat comme suit :

```

decimal montant = Convert.ToDecimal(txtMontant.Text);
decimal acompte = Convert.ToDecimal(txtAcompte.Text);
if (acompte > montant)
    errMessage.SetError(txtAcompte, "L'acompte ne doit pas dépasser le montant");
else
{
    errMessage.SetError(txtAcompte, "");

    //Affectez les valeurs au contrat
    unContrat.conMontant = montant;
    unContrat.conAcompte = acompte;
    unContrat.conPaye = acompte;
    unContrat.cliNo = Convert.ToDecimal(cboClients.SelectedValue.ToString());
    unContrat.empNo = Convert.ToDecimal(cboVendeurs.SelectedValue.ToString());
    unContrat.voyNo = Convert.ToDecimal(cboVoyages.SelectedValue.ToString());
    this.Close();
}
}
}
}

```

- Exécutez pour vérifier.
- Programmez l'événement *Click* du bouton **btnFermer** sur le formulaire **frmAjouter** comme suit :
 suivante. Ici, le formulaire n'a rien ajouté dans le nouveau contrat


```

private void btnFermer_Click(object sender, EventArgs e)
{
    this.Close();
}

```
- Lorsque le formulaire **frmAjouter** se ferme, le formulaire **frmListeDeValeurs** doit ajouter ce contrat dans la *DataTable* **contrat**. Complétez la programmation du bouton **btnAjouter** du formulaire **frmListeDeValeurs** comme suit :

```

frmAjout.ShowDialog();

if (unContrat.conMontant!=-1) //les données de frmAjouter sont entrées
{
    //ajout d'une entrée dans la table contrat
    bdVoyagesDataSet.contrat.AddcontratRow(unContrat);

    //se positionner sur le contrat qui a été ajouté

    contratBindingSource.MoveLast();
    MessageBox.Show(" le contrat " + unContrat.conNo.ToString() + " a été ajouté.", "Ajout d'un contrat",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
}

```

- Exécutez pour tester.
- Procédez de la même manière pour implanter toutes les opérations avec la présentation tabulaire. Utiliser le formulaire **frmListeDeValeursTab**.

5. Gestion des conflits concurrentiels

- Exécutez l'application (affichage tabulaire). Observez que le montant de l'acompte du contrat 444001 est de 200.00\$.

No de contrat	Date	Acompte	Montant
444001	2013-05-04	200,00	1140,58

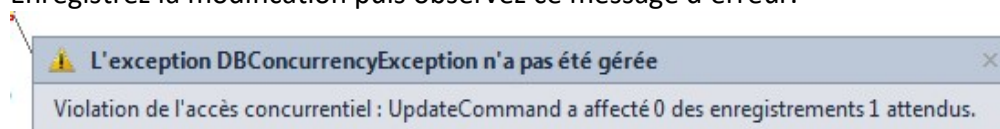
- Sans arrêter l'exécution de l'application, en *Visual Studio*, en passant par l'*Explorateur de serveurs*, modifiez le montant de l'acompte du contrat 444001 pour 100.00\$.

conNo	conDate	conAcompte	conMontant
444001	2013-05-04 00:00...	100,00	1140,58

- Dans le *DataGridView* de l'application, modifiez le montant du contrat 444001 pour 1000.00\$ (ne modifiez pas le montant de l'acompte).

No de contrat	Date	Acompte	Montant
444001	2013-05-04	200,00	1000,00

- Enregistrez la modification puis observez ce message d'erreur.



- Une manière de régler ce problème est d'intercepter l'erreur et d'informer l'utilisateur.

- Reprogrammez l'événement *click* du bouton **btnEnregistrer** de la manière suivante.

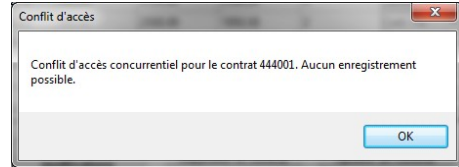
```
private void btnEnregistrer_Click(object sender, EventArgs e)
{
    Validate();
    contratBindingSource.EndEdit();

    try
    {
        tableAdapterManager.UpdateAll(bdVoyagesDataSet);
        MessageBox.Show("Les modifications ont été enregistrées dans la base de données.", "Enregistrement des données");
    }
    catch (DBConcurrencyException erreur)
    {
        String noContrat = erreur.Row["conNo"].ToString();
        MessageBox.Show("Conflit d'accès concurrentiel pour le contrat " + noContrat + ".Aucun enregistrement possible.", "Conflit d'accès");
    }
}
```

- Exécutez l'application. Observez que le montant de l'acompte du contrat 444001 est de 100.00\$.

- Sans arrêter l'exécution de l'application, en *Visual Studio*, en passant par l'*Explorateur de serveurs*, modifiez le montant de l'acompte du contrat 444001 pour 200.00\$.
- Dans le *DataGridView* de l'application, modifiez le montant du contrat 444001 pour 1000.00\$ (ne modifiez pas le montant de l'acompte).

- Enregistrez la modification puis observez ce message d'erreur.



Cette solution est bonne (car elle évite une erreur d'exécution) mais elle n'est pas tout à fait satisfaisante car le conflit d'accès est toujours présent. De plus, la mise à jour s'arrête. Ce qui signifie que toutes les autres modifications (qu'il y ait un conflit d'accès ou non) ne sont pas enregistrées.

- Pour résoudre le conflit d'accès, il faut gérer l'événement *RowUpdated* du *contratTableAdapter*. Cet événement a lieu à la suite de la mise à jour d'un enregistrement dans la table **contrat**. Le gestionnaire d'événement n'est pas directement accessible dans *Visual Studio*. Nous devons l'ajouter par programmation.

- Ajoutez ce qui suit dans la méthode *frmListeDeValeursTab_Load*

```
// Ajouter ce gestionnaire d'événements
contratTableAdapter.Adapter.RowUpdated += new System.Data.SqlClient.SqlRowUpdatedEventHandler(contratTableAdapter_RowUpdated);
```

- Puis programmez cette méthode comme suit : (lisez les commentaires)

```
private void contratTableAdapter_RowUpdated(Object sender, System.Data.SqlClient.SqlRowUpdatedEventArgs e)
{ // Cette méthode est appelée à la suite de la mise à jour d'un enregistrement dans la table contrat

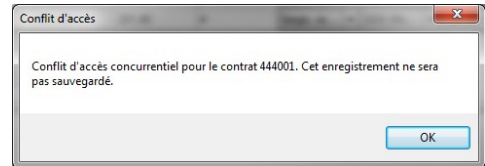
    if (e.RecordsAffected == 0)
    { // L'enregistrement n'a pas été mis à jour dans la table contrat.
      // Il y a eu un conflit d'accès.
      String noContrat = e.Row["conNo"].ToString();
      MessageBox.Show("Conflit d'accès concurrentiel pour le contrat " + noContrat + ". Cet enregistrement ne sera pas sauvegardé.", "Conflit d'accès");

      // Pour poursuivre la mise à jour des enregistrements suivants
      e.Status = UpdateStatus.Continue;
    }
}
```

- Enlevez la modification que vous avez faite dans l'événement *click* du bouton de commande **btnEnregistrer** (enlevez le *try..catch*).
- Exécutez l'application. Observez que le montant de l'acompte du contrat 444001 est de 200.00\$.

- Sans arrêter l'exécution de l'application, en *Visual Studio*, en passant par l'*Explorateur de serveurs*, modifiez le montant de l'acompte du contrat 444001 pour 100.00\$.

- Dans le *DataGridView* de l'application, modifiez le montant du contrat 444001 pour 1000.00\$ (ne modifiez pas le montant de l'acompte).



- Enregistrez la modification puis observez ce message d'erreur. On vous signale que cet enregistrement ne sera pas sauvegardé. Par contre, les autres enregistrements qui ont été modifiés (s'il y en a) devraient être sauvegardés.
- Bien comprendre les autres possibilités offertes pour la gestion des conflits (voir les autres valeurs possibles pour e.status)

6. Enregistrer les données du *DataSet* dans le format *XML*

- Il est possible d'enregistrer les données présentes dans une *DataTable* dans le format *XML*.

- Ajoutez un bouton **btnEnregistrerEnXML** en bas du formulaire **frmListeDeValeursTab**.

Enregistrer les contrats
dans le format XML

- Programmez le *click* de ce bouton comme suit :

```
private void btnEnregistrerEnXML_Click(object sender, EventArgs e)
{
    bDVoyagesDataSet.contrat.WriteXml("Contrats.xml");
}
```

- Exécutez l'application puis cliquez sur ce bouton.
- Observez, dans votre dossier **Debug**, que tous les contrats ont été enregistrés dans le format *XML* (**Contrats.xml**).

```

<?xml version="1.0" standalone="true"?>
<BDVoyagesDataSet xmlns="http://tempuri.org/BDVoyagesDataSet.xsd">
  - <contrat>
    <conNo>444001</conNo>
    <conDate>2014-05-04T00:00:00-04:00</conDate>
    <conAcompte>200.00</conAcompte>
    <conMontant>1140.58</conMontant>
    <ConPaye>200.00</ConPaye>
    <conTypeOcc>3</conTypeOcc>
    <empNo>1006</empNo>
    <voyNo>3004</voyNo>
    <cliNo>123006</cliNo>
  </contrat>

```

➤ Il est possible d'enregistrer son schéma dans un fichier **.xsd**.

- Programmez l'événement *click* du bouton de la manière suivante.

```

private void btnEnregistrerEnXML_Click(object sender, EventArgs e)
{
    bDVoyagesDataSet.contrat.WriteXml("Contrats.xml");
    bDVoyagesDataSet.contrat.WriteXmlSchema("Contrats.xsd");
}

```

- Exécutez l'application puis cliquez sur ce bouton.
- Observez, dans votre dossier **Debug**, que le schéma de la table **contrat** a été enregistré dans le format *XML* dans un fichier **.xsd** (**Contrats.xsd**). Ouvrez-le avec un éditeur texte.