Laboratoire 9
LINQ To SQL – Partie 1

1. Objectifs d'apprentissage

- Utiliser *LINQ to SQL*
- Comprendre ce qu'est un *DataContext*
- Comprendre la conversion des enregistrements en entités
- Comprendre les associations entre les entités
- Comprendre et utiliser les propriétés de navigation des entités
- Lier une requête *LINQ* à un contrôleur
- Enregistrer des modifications dans une base de données
- Effectuer des modifications par programmation
- Gérer les conflits d'accès
- Visualiser les requêtes *SQL* envoyées par le *DataContext*

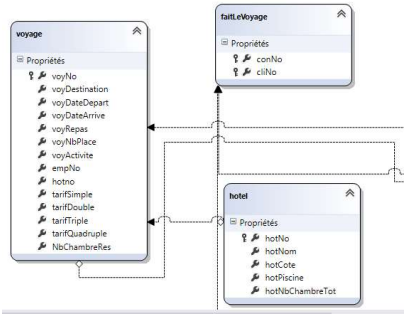
2. Le concepteur Objet/Relationnel

➢ Créez un nouveau projet labo9 de type *Application Windows Forms Viusal C#*.

Renommez votre formulaire **frmLabo9.cs**. Le titre du formulaire doit être *Labo 9 (VotreNom)*.

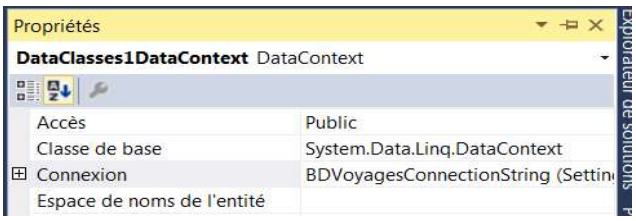
- Ajoutez un composant *Classes LINQ to SQL* à votre projet.

- À partir de l'*Explorateur de serveurs*, glissez toutes les tables de votre base de données **BDVoyagesVotreNom** dans la partie gauche du *Concepteur Objet/Relationnel*. Vous devriez voir ce schéma (dans un fichier *.dbml*).



Le schéma que vous voyez présentement, c'est le schéma de votre base de données <u>en tant qu'***entités***</u>
Enregistrer ce fichier.

3. Le *DataContext*

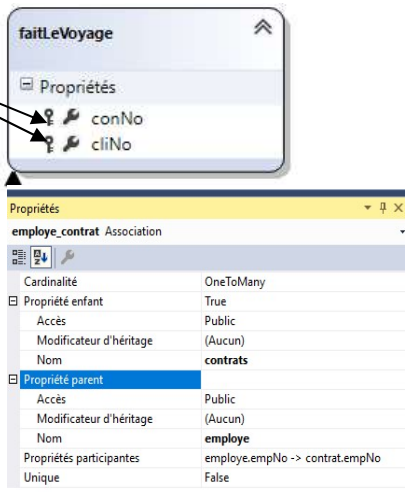


En faisant cela, vous avez créé une classe de type *DataContext*. Cette *classe* est très importante. Elle contient, autres autres, votre chaîne de connexion ainsi que tous les enregistrements de votre base de données qui seront transformés, plus tard, en ***entités***.

Le *DataContext* est à la base de *LINQ To SQL*. Le *DataContext* fait le lien entre les enregistrements de votre base de données réelle et les entités utilisables par *LINQ*.

Lorsque vous exécutez une requête *LINQ*, le *DataContext* crée une requête *SQL* qui est exécutée sur la base de données. Le résultat de la requête *SQL* est transmis au *DataContext* qui le convertit en entités utilisables par *LINQ*.

4. Les entités et les associations



Observez que chaque entité a des des propriétés (qui sont en fait les champs de la table correspondante dans la base de données).

Maintenant, cliquez sur le lien qu'il y a entre la table **employe** et la table **contrat**. Ce lien (qui porte le nom de **employe_contrat**) est une ***association***. Il associe des entités de la table **employe** à des entités de la classe **contrat** (et vice et versa).

Dans la table **employe** et dans la table **contrat**, il y a des propriétés que vous ne voyez pas présentement sur le schéma. Ces propriétés portent le nom de ***propriétés de navigation*** car vous pouvez les utiliser pour naviguer d'un type d'entité à un autre.

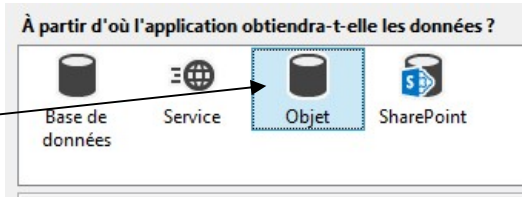
Obervez, dans l'association **employe_contrat**, la propriété enfant qui porte le nom de **contrats**. Cette propriété est située dans la table **employe** et elle contient la liste de tous les contrats qui ont été associés à cet employé. Par exemple, l'employé 1001 n'a aucun contrat; par conséquent, cette propriété contient une liste vide. L'employé 1003 a 6 contrats, cette propriété contient ces 6 contrats.

Observez, dans l'association **employe_contrat**, la propriété parent qui porte le nom de **employe**. Cette propriété est située dans la table **contrat** et elle contient un employé particulier. Par exemple, le contrat 444001 est détenu par l'employé 1006. Par conséquent, cette propriété contient l'employé 1006.

Ce qu'il faut comprendre ici, c'est que les clefs ne sont plus réellement nécessaires. On peut naviguer d'un type d'entités à un autre en utilisant seulement les propriétés de navigation.

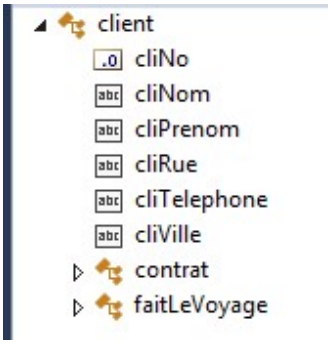
5. Création d'une source de données à l'aide de l'Assistant

- Maintenant, ajoutez une nouvelle *Source de données*. Étant donné que *LINQ* utilise des objets et non pas une base de données, choisissez *Objet*.



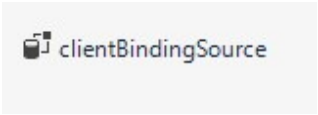
- Puis choisissez l'objet **client** (cet objet représente un **client** particulier de votre base de données).

Observez les propriétés **contrat** et **faitLeVoyage** dans la *Source de données***.** Ce sont des propriétés de navigation. La propriété **contrat** contient la liste de tous les contrats d'un client et la propriété **faitLeVoyage** contient la liste de toutes les entités situées dans la table **faitLeVoyage** qui sont associées à ce client.



- Sur votre formulaire, ajoutez un onglet dont le titre *Gestion des clients*.

- À partir de la fenêtre *Source de données*, glissez, sur cet onglet, l'objet **client** sous la forme tabulaire (dans un *DataGridView*) puis supprimez complètement la barre de navigation (**clientBindingNavigator**).



Observez que l'objet **clientBindingSource** a automatiquement été ajouté. Cet objet est le contrôleur. Il fait le lien entre les vues et votre source de données (ici, la source de données n'est pas un *DataSet* mais les enregistrements de la table **client** qui ont été transformés en entités).

6. Utilisation d'un DataContext et d'un contrôleur

Le *DataContext* est très important car c'est lui qui fait le lien entre les objets utilisables par *LINQ* (les entités) et les tables de votre base de données.

- Instanciez un nouveau *DataContext* dans la classe **frmLabo9**.

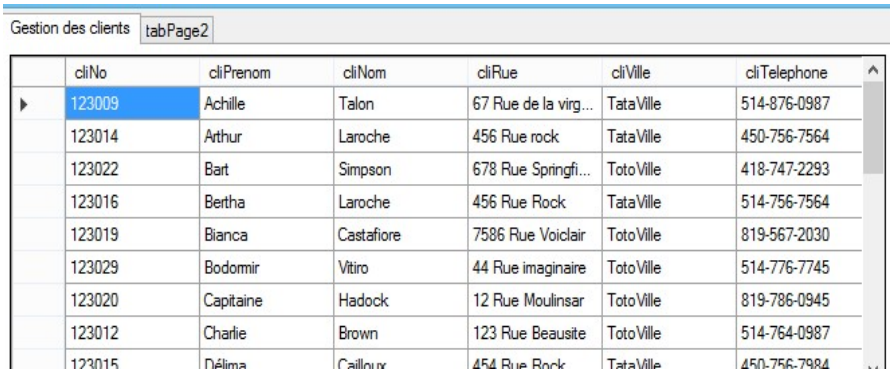
```
public partial class frmLabo9 : Form
{

    DataClasses1DataContext monDataContext = new DataClasses1DataContext();
```

- Puis programmez l'événement *Load* du formulaire de la manière suivante :

```
clientBindingSource.DataSource =
    from unClient in monDataContext.client
    let nomCompletClient = unClient.cliPrenom + " " + unClient.cliNom
    orderby nomCompletClient
    select unClient;
```

- Exécutez le programme et observez que vous voyez, dans le *DataGridView*, tous les enregistrements de la table **client** dans la vraie base de données.

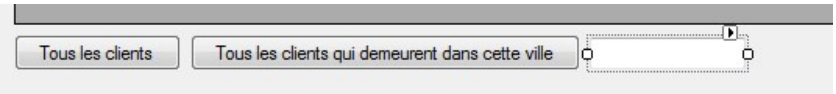


Ici, le *DataContext* récupère tous les enregistrements de la table **client** de la base de données puis les place dans ***monDataContext.client***

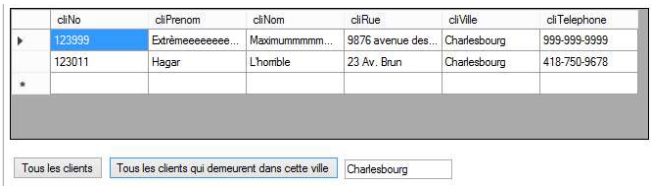
C'est le contrôleur qui a exécuté votre requête *LINQ*.

Étant donné que le *DataGridView* est lié à ce contrôleur, alors vous voyez, dans le *DataGridView*, tous les clients qui ont été sélectionnés par votre requête *LINQ*).

➢ Ajoutez, en bas du *DataGridView*, deux boutons: *Tous les clients* et *Tous les clients qui demeurent dans cette ville* suivis d'une zone de texte (txtVille). Nommer les boutons **btnTousLesClients** et **btnTousLesClientsParVille** respectivement.



- Le bouton *Tous les clients*, permet d'afficher tous les clients <u>en ordre de nom complet du client</u> (comme au chargement) et, *Tous les clients qui demeurent dans cette ville*, permet d'afficher seulement les clients qui demeurent dans la ville saisie dans la zone de texte. L'affichage est toujours <u>en ordre de nom complet du client</u>.



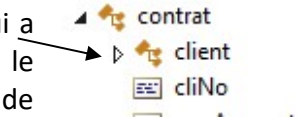
- Il est important de prendre note, qu'à chaque fois qu'une requête *LINQ* est exécutée, que le *DataContext* va chercher les données directement dans la vraie base de données. Pour illustrer cela, exécutez l'application, puis, <u>sans fermer l'application</u>, dans l'*Explorateur de serveurs*, ajoutez un nouveau client.

- Maintenant, dans votre application, cliquez sur le bouton *Tous les clients* et observez que ce client est présent.

7. Utilisation des propriétés de navigation

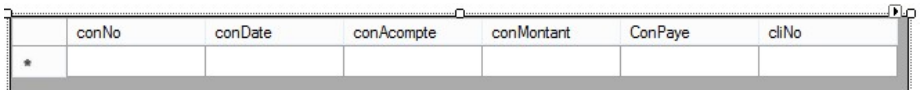
➢ Ajoutez l'objet **contrat** comme nouvelle *Source de données*.

Observez, dans cet objet, qu'il y a une propriété qui porte le nom de **client**. Cette propriété contient l'entité **client** qui a été associée à un contrat donné (c'est le client dont le numéro est **cliNo**). Il s'agit donc d'une propriété de navigation.



employe, voyage et faitLeVoyage sont aussi des propriétés de navigation.

- Maintenant, à partir de la fenêtre *Source de données*, glissez l'objet **contrat** sur l'onglet sous la forme tabulaire (dans un *DataGridView*). Supprimez toutes les colonnes sauf *conNo*, *conDate*, *conAcompte, conMontant*, *conPaye* et *cliNo*.

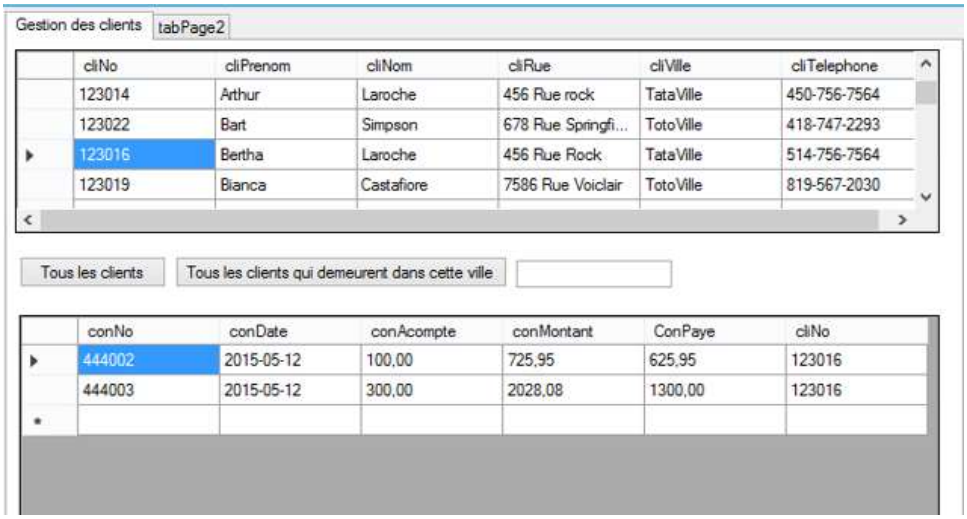


- Pour afficher tous les contrats du client sélectionné, vous devez programmer l'événement *CurrentChanged* du contrôleur *clientBindingSource* comme suit :

```
private void clientBindingSource_CurrentChanged(object sender, EventArgs e)
{
    if (clientBindingSource.Current != null)
    {
        contratBindingSource.DataSource =
            from unContrat in monDataContext.contrat
            where unContrat.client == (client)clientBindingSource.Current
            select unContrat;
     }
}
```

Notez l'utilisation de la propriété de navigation **client** de l'objet **contrat**.

- Exécutez l'application et sélectionnez un client quelconque.



Nous aurions pu également utiliser la propriété de navigation **contrat** qui est située dans un entité **client**. Cette propriété contient la liste de tous les contrats qui ont été associés à un client donné. Par conséquent, nous aurions pu programmer l'événement précédent de la manière suivante. Cela aurait donné les mêmes résultats.

```
private void clientBindingSource_CurrentChanged(object sender, EventArgs e)
 {
    if (clientBindingSource.Current != null)
     {
         contratBindingSource.DataSource =
             ((client)clientBindingSource.Current).contrat;
     }
 }
```

8. L'enregistrement des modifications dans la base de données

➢ Ajoutez, en bas du *DataGridView* des clients, un bouton *Enregistrer*.

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

```
private void cmdEnregistrer_Click(object sender, EventArgs e)
{
    clientBindingSource.EndEdit();
    contratBindingSource.EndEdit();

    monDataContext.SubmitChanges();
}
```

La méthode *SubmitChanges* du *DataContext* est une méthode très importante car c'est elle qui met à jour la base de données et ce, à partir des entités créées par *LINQ To SQL*.

- Exécutez l'application puis faites les modifications suivantes :

  1- Au niveau des clients, modifiez le nom de *Ned Triton* par *Led Trito*

  2- Dans le *DataGridView*, ajoutez le client suivant :

	123400	Michel	Jacques	125 Rue Imaginaire	TotoVille	666-666-6666

  3- Cliquez sur le bouton **Enregistrer**.

- Cliquez sur le bouton *Tous les clients* pour vérifier. Ce bouton recharge toutes les données de la base de données.

Le *DataContext* garde la trace, dans un objet de type ***ChangeSet***, de la liste de toutes les modifications qui doivent être effectuées dans la base de données (mises à jour, suppressions et insertions). L'appel à la méthode *SubmitChanges* enregistre ces modifications. Cette liste devient vide dès que l'enregistrement est terminé. Il est possible d'obtenir cette liste en faisant appel à la méthode *GetChangeSet* du *DataContext*.

Il est également important de noter que ce sont toutes les modifications du *DataContext* qui sont enregistrées. Par conséquent, si vous faites des modifications sur les contrats, ces modifications vont également être enregistrées.

9. La gestion des erreurs lors de l'enregistrement des modifications

Lorsque l'utilisateur effectue des modifications au niveau des entités, toutes les validations ne sont pas effectuées. L'enregistrement des modifications peut donc échouer.

Par exemple, si l'utilisateur supprime un client qui a un contrat et qu'il enregistre cette modification dans la base de données, cela ne fonctionnera pas (problème d'intégrité référentielle).

➢ Pour l'instant, nous allons gérer ces types d'erreur à l'aide d'un **try…catch**. Modifiez la programmation de l'événement *Click* du bouton **btnEnregistrer**.

```
private void btEnregistrer_Click(object sender, EventArgs e)
{
    clientBindingSource.EndEdit();
    contratBindingSource.EndEdit();

    try {
        monDataContext.SubmitChanges();
        }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, " Erreur lors de l'enregistrement des données");
        btnTousLesClients.PerformClick();
    }
}
```

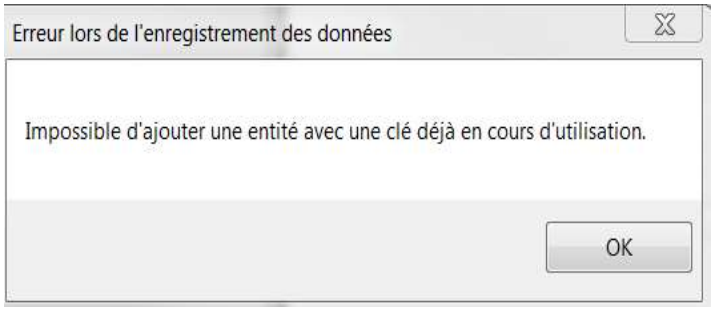
S'il y a une erreur lors de l'enregistrement des modifications dans la base de données, nous affichons un message d'erreur puis nous appelons l'événement *Click* du bouton **btnTousLesClients** pour recharger tous les clients de la base de données.

- Exécutez l'application et ajoutez le contrat 444005 pour le client 123007.

	conNo	conDate	conAcompte	conMontant	ConPaye	cliNo
	444013	2015-12-13	342,70	342,70	0,00	123007
	444005	2016-10-25	123	122	10	123007

Comme vous pouvez le constater, au niveau des entités, l'ajout n'a causé aucun problème.

- Cliquez sur le bouton **Enregistrer**. Observez le message d'erreur. L'erreur provient du fait que le contrat 444005 existe déjà.



Pour éviter l'erreur, toutes les validations qui s'imposent doivent être effectuées avant l'enregistrement dans la base de données. Ceci n'est pas le but de ce laboratoire.

10. Effectuer des modifications par programmation

Normalement, les insertions, les suppressions et les mises à jour d'entités sont réalisées de manière interactive à l'aide du *DataGridView* et/ou de la barre de navigation. Mais il est possible de le faire par programmation.

Voici comment insérer, supprimer et mettre à jour une entité par programmation à l'aide de *LINQ To SQL*.

→ **Exemple d'insertion du client 123401 par programmation**

```
client unClient = new client
{
    cliNo = 123401,
    cliPrenom = "Michel",
    cliNom = "L'Heureux",
    cliRue = "125 rue Papineau",
    cliVille = "Montréal",
    cliTelephone = "999-999-9999"
};
clientBindingSource.Add(unClient);
```

Notez qu'ici, nous avons ajouté le client 123401 en utilisant l'instruction ***clientBindingSource.Add(unClient)*** mais nous aurions pu également utiliser l'instruction ***monDataContext.client.InsertOnSubmit(unClient)***.

La première instruction ajoute le client en tant qu'entité dans le modèle ainsi que dans la vue (dans le *DataGridView*). La seconde ajoute le client dans le modèle mais pas dans la vue (dans le *DataGridView*).

→ **Exemple de suppression du client 123401 par programmation**

```
var clientASupprimer = from unClient in monDataContext.client
                       where unClient.cliNo == 123401
                       select unClient;

foreach (var unClient in clientASupprimer)
    clientBindingSource.Remove(unClient);
```

Noter que ***clientBindingSource.Remove(unClient)*** peut être remplacée par l'instruction ***monDataContext.client.DeleteOnSubmit(unClient)***.

→ **Exemple de mise à jour du nom de la ville du client 123401 par programmation**

```
var clientAMettreAJour = from unClient in monDataContext.client
                         where unClient.cliNo == 123401
                         select unClient;

foreach (var unClient in clientAMettreAJour)
    unClient.cliVille = "Montréal";
```

Ici, nous allons effectuer une mise à jour par programmation.

➢ Sur l'onglet, ajoutez deux zones de texte (**txtVilleSource** et **txtVilleCible**) et un bouton *Remplacer cette ville par celle-ci* (**btnRemplacerVille**).

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

```
private void btnRemplacerVille_Click(object sender, EventArgs e)
{
    var requeteRemplacerVille = from unClient in monDataContext.client
                                where unClient.cliVille.Trim() == txtVilleSource.Text.Trim()
                                select unClient;

    foreach (var unClient in requeteRemplacerVille)
        unClient.cliVille = txtVilleCible.Text.Trim();
}
```

- Exécutez le programme et remplacez la ville *TotoVille* par *TitiVille* en utilisant le bouton *Remplacer cette ville par celle-ci.*

Observez, dans le *DataGridView*, que tous les *TotoVille* ont été remplacés par des *TitiVille*.

cliNo	cliPrenom	cliNom	cliRue	cliVille	cliTelephone
123030	Aaaaa	Bbbbb	1 rue Inconnu	TitiVille	888-888-8888
123009	Achille	Talon	67 Rue de la virg...	TataVille	514-876-0987
123014	Arthur	Laroche	456 Rue rock	TataVille	450-756-7564
123022	Bart	Simpson	678 Rue Springfi...	TitiVille	418-747-2293

- Ces mises à jour ont été faites au niveau des entités **client** mais n'ont pas été faites dans la vraie base de données. Pour vous en convaincre, fermez l'application puis ouvrez-la de nouveau. Observez que la ville est *TotoVille* (et non pas *TitiVille*).

  Pour que ces mises à jour soient enregistrées dans la vraie base de données, il aurait fallu cliquer sur le bouton **Enregistrer** ou utiliser l'instruction suivante à la fin de l'événement *Click* du bouton.

```
monDataContext.SubmitChanges();
```

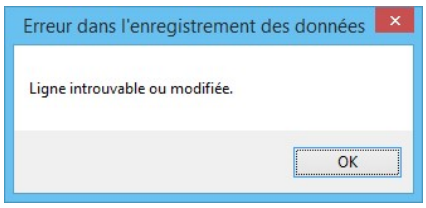
11. Les conflits d'accès

➢ Ré-exécutez le programme.

- Sans fermer l'exécution du programme, en passant par l'*Explorateur de serveurs*, modifiez le prénom du client 123009.

- Dans le *DataGridView* des clients, modifiez le nom du client 123009 puis cliquez sur le bouton **Enregistrer**.

  Observez cette boîte de dialogue. Il y a un conflit d'accès. Le prénom du client a été modifié dans la base de données mais pas dans l'entité **client**. Le prénom du client 123009 dans la table **client** ne correspond pas au prénom du client 123009 qui a été chargé dans l'entité **client**.



  Ici, le *DataContext* a adopté la solution optimiste pour éviter les conflits d'accès c'est-à-dire, lors du chargement, il a fait une copie de sécurité de l'enregistrement original. Avant l'enregistrement des modifications, le *DataContext* a rechargé le même enregistrement et il s'est aperçu que cet enregistrement était différent de l'enregistrement original. C'est ce qui a engendré le conflit d'accès.

- Voici une solution au problème. Programmez-la (ne pas oublier de taper *using System.Data.Linq*) :

```
private void btEnregistrer_Click(object sender, EventArgs e)
{
    clientBindingSource.EndEdit();
    contratBindingSource.EndEdit();

    try {
        monDataContext.SubmitChanges(ConflictMode.ContinueOnConflict);
        }
    catch (ChangeConflictException )
    {
        monDataContext.ChangeConflicts.ResolveAll(RefreshMode.KeepCurrentValues);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, " Erreur lors de l'enregistrement des données");
        btnTousLesClients.PerformClick();
    }
}
```

Explications:

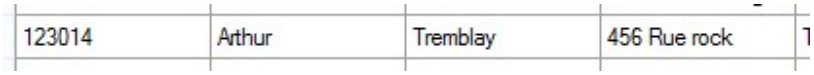
Le *submitChanges(ConflictMode.ContinueOnConflict)* permet de poursuivre l'enregistrement des modifications même s'il existe des conflits d'accès. Dans ce mode, toutes les modifications qui ne présentent pas de conflits d'accès sont enregistrées normalement et celles qui présentent des conflits d'accès sont placées dans une liste spéciale qui porte le nom de *ChangeConflicts*.

Le *ChangeConflicts.ResolveAll(RefreshMode.KeepCurrentValues)* permet de résoudre tous les conflits d'accès qui ont été placés dans la liste *ChangeConflict* en enregistrant seulement les valeurs qui ont été modifiées par l'utilisateur courant. Les modifications effectuées par d'autres utilisateurs, après le chargement des données, seront ignorées.

- Maintenant, exécutez le programme.

- Sans fermer l'exécution du programme, en passant par l'*Explorateur de serveurs*, modifiez le prénom du client 123014; donnez-lui le prénom de ***Ti-Coune***.

- Dans le *DataGridView* des clients, modifiez le nom du client 123014; donnez-lui le nom de ***Tremblay*** (ne modifiez pas son prénom).



- Cliquez sur le bouton **Enregistrer** puis sélectionnez l'enregistrement 123014. Observer les modifications effectuées.

Ici, nous avons adopté cette solution pour résoudre les conflits d'accès, mais il en existe deux autres :

→ *RefreshMode.OverwriteCurrentValues* **:** bien comprendre cette option et tester

→ *RefreshMode.KeepChanges* **:** bien comprendre cette option et tester

12. Les requêtes *SQL* transmises par le *DataContext*

Le *DataContext* transmet des requêtes *SQL* à la base de données pour charger les enregistrements en mémoire (dans le but de les convertir en entités) et pour enregistrer les modifications de l'utilisateur base de données. Il est possible de visualiser les requêtes *SQL* qui sont transmises à la base de données réelle par le *DataContext*.

- Pour ce faire, au début de l'événement *Load* du formulaire, ajoutez ce qui suit (ne modifiez pas ce qui a déjà été programmé).

```
monDataContext.Log = new System.IO.StreamWriter("Linq-To-SQL.log");
```

Toutes les requêtes *SQL* transmises par le *DataContext* seront sauvegardées automatiquement dans le fichier **Linq-To-SQL.log**.

- Dans l'événement *FormClosed* du formulaire (qui est appelé à la fin de la fermeture du formulaire), tapez ce qui suit :

```
monDataContext.Log.Close();
```

- Exécutez le programme puis effectuez des opérations quelconques puis fermez le formulaire.

- Maintenant, regardez le contenu du fichier **Linq-To-SQL.log** (situé dans le dossier **Labo9\bin\Debug**). Vous pouvez visualiser toutes les requêtes *SQL* qui ont été transmises à la base de données depuis l'ouverture du formulaire jusqu'à sa fermeture.