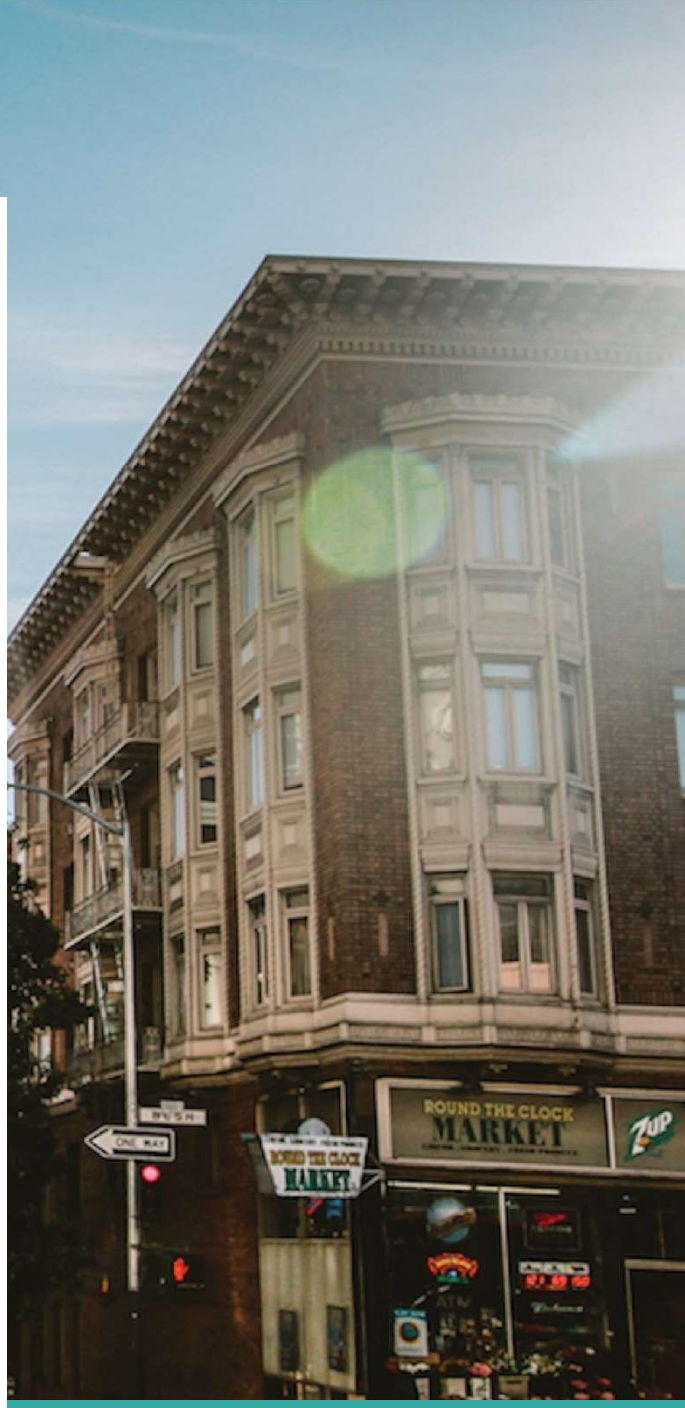
VBNET & BD en Mode connecté



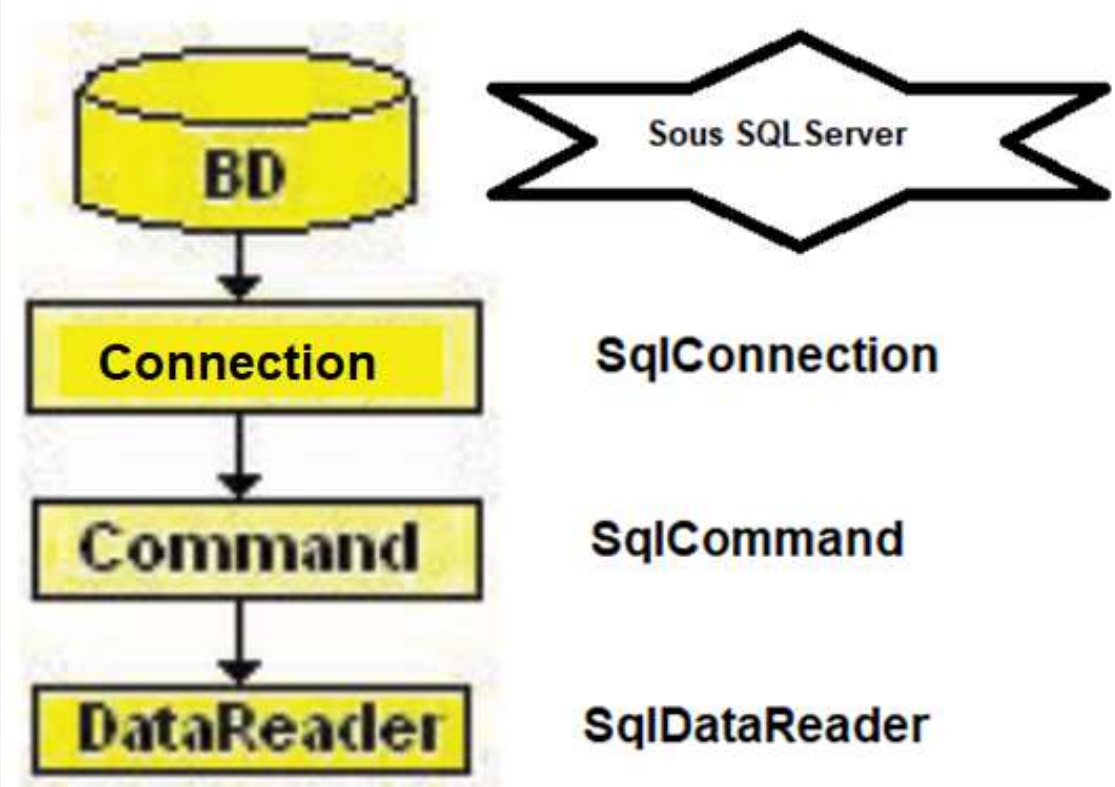
BTS DSI 2

Nom complet
Créé par : Mourad BOUAABID

Généralités

Un objet **DataReader** fournit des données en lecture seule en un temps record. La seule possibilité est de se déplacer en avant.
En contrepartie de sa rapidité il monopolise la connexion.
Il faut créer un objet **Connection** puis un objet **Command**, ensuite on exécute la méthode **ExecuteReader** pour créer l'objet **DataReader**; enfin on parcourt les enregistrements avec la méthode **Read**.

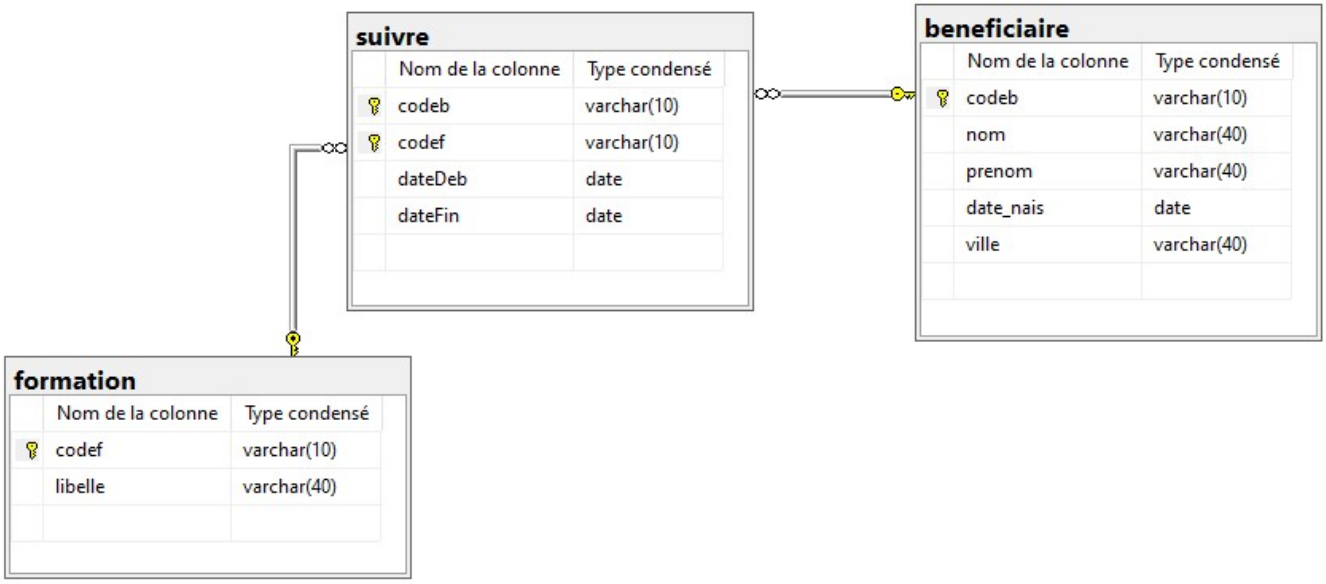
L'architecture des objets de communication avec une base de données sous **SQL Server** en mode connecté :



Il faut importer les **NameSpaces** nécessaires.
Imports System.Data.SqlClient

Etude de cas : Gestion des Formations

Le MLDR de la base de données de cette gestion est donné ci-après :



Les caractéristiques du serveur de base de données sont :

- Nom de la base de données : **db_Formation**
- Nom du serveur : **Server1**
- Authentification : **celle du Windows**.

On veut créer une application Desktop permettant :

- La gestion des bénéficiaires ;
- La gestion des inscriptions.

Notre application sera constituée :

- Un module contenant les variables globales de notre projet ;
- Un Form pour la gestion des bénéficiaires ;
- Un Form pour la gestion des inscriptions.

Etablir une connexion avec BD

Dans un module, on déclare le premier objet qui est celui de connexion avec la base de données.

```
Public con  As SqlConnection = Nothing
```

Dans le même module, on crée une procédure de connexion que l'on nomme : **Connecter**
Afin d'établir une connexion avec la base de données, on suit les étapes suivantes :

- On instancie l'objet de connexion, lors cette instanciation, on envoie au constructeur une chaine de connexion :
  "Initial Catalog = db_Formation ; Data Source = Server1 ; Integrated Security= true ;"
- On appelle la méthode d'instance : **open**

```
Public Sub Connecter()
con = New SqlConnection("Initial Catalog = db_Formation ; Data Source =
Server1 ; Integrated Security= true ;")
con.open()
End Sub
```

Pour gérer les éventuelles erreurs causées par l'échec de connexion, on utilise les exceptions.
Si on veut afficher, dans une boîte de dialogue, un message d'erreur en cas d'échec de connexion, on procède comme suit :

```
Public Sub Connecter()
Try
con = New SqlConnection("Initial Catalog = db_Formation ; Data Source =
Server1 ; Integrated Security= true ;")
con.open()
Catch
  MessageBox.Show("Echec de connexion")
End Try
End Sub
```

Le message affiché dans la boîte de dialogue est personnalisé, si on veut afficher le message généré par VBNET, on exploite l'objet de type **Exception**, cet objet possède

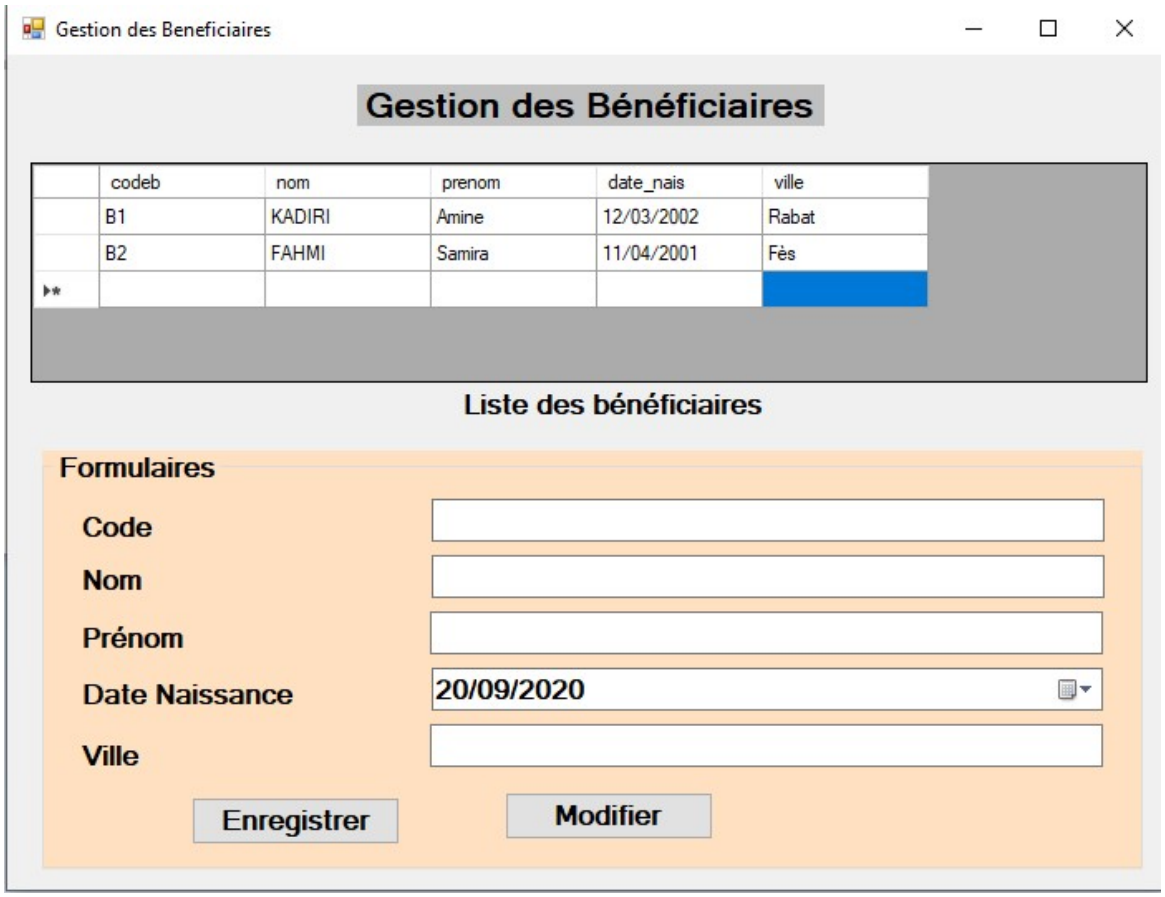
d'une propriété **Message** qui contient le message d'erreur. La procédure **Connecter** devient :

```
Public Sub Connecter()
Try
con = New SqlConnection("Initial Catalog = db_Formation ; Data Source =
Server1 ; Integrated Security= true ;")
con.open()
Catch Ex As Exception
  MessageBox.Show(Ex.Message)
End Try
End Sub
```

Gestion des bénéficiaires

Afin de gérer les bénéficiaires, on crée une interface adéquate à cette tâche :

.



Les contrôles de cette interface et leurs propriétés sont donnés ci-après :

Contrôle	Propriété	Valeur
Button	Name Text	cmdEnregistrer Enregistrer
	Name Text	cmdModifier Modifier
TextBox	Name	txtCode
	Name	txtNom
	Name	txtPrenom
	Name	txtVille
DateTimePicker	Name Format	dtpDateNais Short
DataGridView	Name ReadOnly	dgvListe True

- **Button** : Un contrôle qui est sensible au clic permettant de déclencher un traitement.
- **TextBox :** Zone d'affichage et aussi de saisie. Il possède plusieurs méthodes et propriétés :

Propriéte/Méthode	Rôle
Text	Propriété pour lire ou écrire
Focus()	Méthode pour positionner le focus sur ce contrôle
Clear()	Vider la zone txete

- **DataGridView :** Une grille d'affichage des données sous-forme de table :

Propriéte/Méthode	Rôle
DataSource	Propriété pour préciser la source de données qui peut être : ➢ Table statique ➢ Table dynamique ➢ DataTable
ReadOnly	Propriété booléenne, s'elle vaut True, on interdit la saisie sur la grille.
CurrentRow	Propriété à lecture seule de type **DataGridViewRow** donne la ligne contenant la cellule active.
CurrentRow.Index	Propriété à lecture seule de type **Integer** donne l'index de la ligne contenant la cellule active.
Rows(index)	Propriété à lecture seule qui retourne la ligne ayant l'index **index**.

Rows(index).Cells(colonne)	Propriété à lecture seule donne la cellule ayant le numéro de colonne **colonne.**
Rows(index).Cells(colonne).value	Propriété à lecture seule qui donne le contenu de la cellule de type **Object**.

- **DateTimePicker :** Un contrôle qui fournit la date :

Propriéte/Méthode	Rôle
Value	Propriété pour lire ou définir une date de type **Date**.
Format	Propriété qui définit la forme de la date : ➢ Short : format court **jj/mm/aaaa** ➢ Long : on indique le jour et le mois sous-forme de String. ➢ Time : l'heure qui s'affiche

Objet Command : SqlCommand

SqlCommand : utilisé pour gérer les requêtes destinées au SGBDR SQL Server.

1. **Préparer l'objet Command** :

Il existe 2 manières pour préparer cet objet :

Méthode 1 :

```
Dim  cmd As SqlCommand = Nothing
cmd = con.CreateCommand()
cmd.CommandType = CommandType.Text
cmd.CommandText = "Requête SQL"
```

Méthode 2 :

```
Dim  cmd As SqlCommand = Nothing
cmd = New SqlCommand("Requête SQL", con)
ou
Dim cmd As New SqlCommand("Requête SQL", con)
```

Remarque :

```
cmd.CommandType = CommandType.Text
```

La propriété **CommandType** précise le type de requête à envoyer au serveur de base de données, les valeurs prises par cette propriété sont :

- **CommandType.Text** : Pour une requête SQL
- **CommandType.StoredProcedure** : Pour l'appel d'une fonction ou procédure stockée.
- **CommandType.TableDirect** : Pour une table

Cette propriété a comme valeur par défaut : **CommandType.Text**

2. **Envoi d'une requête** :

L'objet Command est responsable d'envoi des requêtes, le tableau suivant illustre les commandes adéquates à des cas divers :

Type de requête	**Commande**
Lecture d'une table ou jointure	**Cmd.ExecuteReader()** Retourne un objet **SqlDataReader**
Lecture d'un scalaire (une valeur) ❖ Select max(champ) ❖ **Select min(champ)** ❖ **Select avg(champ)** ❖ **Select count(champ)**	**Cmd.ExecuteScalar**() Retourne une valeur de type **Object**
Les requêtes de mise à jour : ❖ **Insert** ❖ **Delete** ❖ **Update**	**Cmd.ExecuteNonQuery**() Retourne Integer qui est le nombre de lignes touchées par la requête.

3. **L'objet SqlDataReader** :

L'objet SqlDataReader contient les enregistrements prévenant de la base de données.

Cet objet possède un ensemble de propriétés et de méthodes :

Propriéte/Méthode	**Rôle**
Read()	Méthode positionne le cursor sur l'enregistrement suivant s'il existe, elle retourne True sinon, elle retourne False.
GetBoolean(numéro de champ)	Accesseur à un champ booléen
GetString(numéro de champ)	Accesseur à un champ Texte
GetDateTime(numéro de champ)	Accesseur à un champ date
GetInt32(numéro de champ)	Accesseur à un champ entier
GetDouble(numéro de champ)	Accesseur à un champ réel double
GetFloat(numéro de champ)	Accesseur à un champ réel float
(numéro de champ) **Ou GetValue(numéro de champ)**	Accesseur à un champ de type quelconque sous-forme d'object
GetName(numéro de champ)	Retourne le nom du champ correspondant à son numéro.

GetOrdinal(nom de champ)	Retourne le nom de champ.

Remarque :

L'objet **SqlDataReader** monopolise la connexion, donc une fois on termine l'exploitation de cet objet, on doit le fermer via la commande : **Close()**

Si on ignore cette étape, on ne pourra plus communiquer avec la base de données.

Codage de l'interface de gestion des bénéficiaires

➢ Procédure d'affichage des bénéficiaires : **ChargerBen**

```
Private Sub  ChargerBen()
...
End Sub
```

Les étapes à suivre :

- ❖ Préparer un objet **SqlCommand** ;
- ❖ Envoyer une requête de lecture de tous les bénéficiaires ;
- ❖ Préparer un objet **DataTable** ;
- ❖ Charger ce **DataTable** depuis **SqlDataReader** ;
- ❖ Fermer l'objet **SqlDataReader** ;
- ❖ Assigner à la propriété **DataSource** l'objet **DataTable**.

➢ Fonction de vérification de validité du formulaire : **FormValide**

```
Private Function  FormValide() As Boolean
  If txtCode.Text.Trim.Equals("") Or txtNom.Text.Trim.Equals("") Or
txtPrenom.Text.Trim.Equals("") Or txtVille.Text.Trim.Equals("") Then
        Return False
    End If
 Return True
End Function
```

Ce formulaire est valide si toutes les zones texte ne sont pas vides.

➢ Fonction de vérification de l'unicité du code d'un bénéficiaire : **CodeExiste**

```
Private Function  CodeExiste(ByVal c As String) As Boolean
...
End Function
```

Il existe 3 méthodes pour réaliser cette fonction :

Méthode 1 :

Les étapes à suivre :

- Préparer un objet **SqlCommand** ;
- Envoyer une requête, par **ExecuteScalar**, qui lit le nombre de bénéficiaires ayant le code donné en argument ;
- Retourner True ou False si ce nombre lu diffèrent ou égale à 0 ;

```
Private Function  CodeExiste(ByVal c As String) As Boolean

' On prépare une requête
Dim Req As String
Req = "select count(*) from Beneficiaire where CodeB = ' " &  c & " ' "

' On prépare l'objet SqlCommand
Dim cmd As New SqlCommand(Req, con)

' On envoie la requête
Dim N As Integer
N = cmd.ExecuteScalar()

' On retourne True ou False
Return N<>0
End Function
```

Méthode 2 :

Elle ressemble à la méthode précédente mais l'envoi de la requête sera fait par **ExecuteReader** :

```
Private Function  CodeExiste(ByVal c As String) As Boolean

' On prépare une requête
Dim Req As String
Req = "select count(*) from Beneficiaire where CodeB = ' " &  c & " ' "

' On prépare l'objet SqlCommand
Dim cmd As New SqlCommand(Req, con)

' On envoie la requête
Dim rd As SqlDataReader
rd = cmd.ExecuteReader()
```

```
' charger DataTable
Dim T As New DataTable
T.Load(rd)

' On ferme SqlDataReader
Rd.Close()

' On récupère le résultat
Dim N As Integer
N = T.Rows(0).Item(0)

' On retourne True ou False
Return N<>0
End Function
```

Méthode 3 :

Les étapes à suivre :

- Préparer un objet **SqlCommand** ;
- Envoyer une requête, par **ExecuteReader**, qui lit le bénéficiaire ayant le code donné en argument ;
- Préciser l'état de retour selon l'existence ou non d'enregistrement dans **SqlDataReader** ;
- Retourner True ou False si ce nombre lu diffèrent ou égale à 0 ;

```
Private Function  CodeExiste(ByVal c As String) As Boolean

' On prépare une requête
Dim Req As String
Req = "select * from Beneficiaire where CodeB = ' " & c & " ' "

' On prépare l'objet SqlCommand
Dim cmd As New SqlCommand(Req, con)

' On envoie la requête
Dim rd As SqlDataReader
rd = cmd.ExecuteReader()

' On précise la valeur de retour
Dim v As Boolean = rd.Read()
rd.Close()
```

```
' On retourne True ou False
Return v
End Function
```

➢ Procédure d'ajout d'un nouveau bénéficiaire

```
Private Sub addBen(ByVal c As String, ByVal n As String, ByVal p As String, ByVal dn As Date, ByVal v As String)
...
End Sub
```

Les étapes à suivre :

- ❖ Vérifier la validité du formulaire ;
- ❖ Vérifier l'unicité du code bénéficiaire ;
- ❖ Envoyer la requête d'insertion d'un nouveau bénéficiaire.

➢ Procédure d'initialisation du formulaire

```
Private Sub initForm()
...
End Sub
```

Les étapes à suivre :

- ❖ Vider les zones textes
- ❖ Initialiser le **DateTimePicker** par la date système ;
- ❖ Positionner le focus sur la zone texte **txtCode**.

Remarque : En VB, pour récupérer la date système sous-forme d'objet **Date**, on utilise la propriété static **Now** de la classe **DateTime : DateTime.Now**

```
Private Sub initForm()
    txtCode.Clear()
    txtNom.Clear()
    txtPrenom.Clear()
    txtVille.Clear()
    dtpDateNais.Value = DateTime.Now
    txtCode.Focus()
End Sub
```

➢ Procédure événementielle du bouton Enregistrer

```
Private Sub cmdEnregistrer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdEnregistrer.Click

```

```
addBen(txtCode.Text, txtNom.Text, txtPrenom.Text, dtpDateNais.Value,
txtVille.Text)
chargerBen()
initForm()

End Sub
```

- Procédure événementielle de DataGridView : clic sur les données de la grille

  Lorsqu'on clique sur un bénéficiaire, toutes ses informations seront affichées sur le formulaire

```
Private Sub dgvListe_CellContentClick(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.DataGridViewCellEventArgs) Handles
dgvListe.CellContentClick
…
 End Sub
```

Les étapes à suivre :

- Récupérer le numéro de la ligne sélectionnée ;
  - Soit par : **CurrentRow**.**Index** de **DataGridView**
  - Soit par : **e**.**RowIndex** (e : argument de la procédure)
- Récupérer le code du bénéficiaire sélectionné ;
- Envoyer une requête de sélection vers la base de données pour récupérer toutes les informations du bénéficiaire sélectionné ;
- Afficher ces informations sur le formulaire :
  - Soit par : Objet **SqlDataReader**
  - Soit par : Objet **DataTable**

- Procédure de modification d'un bénéficiaire

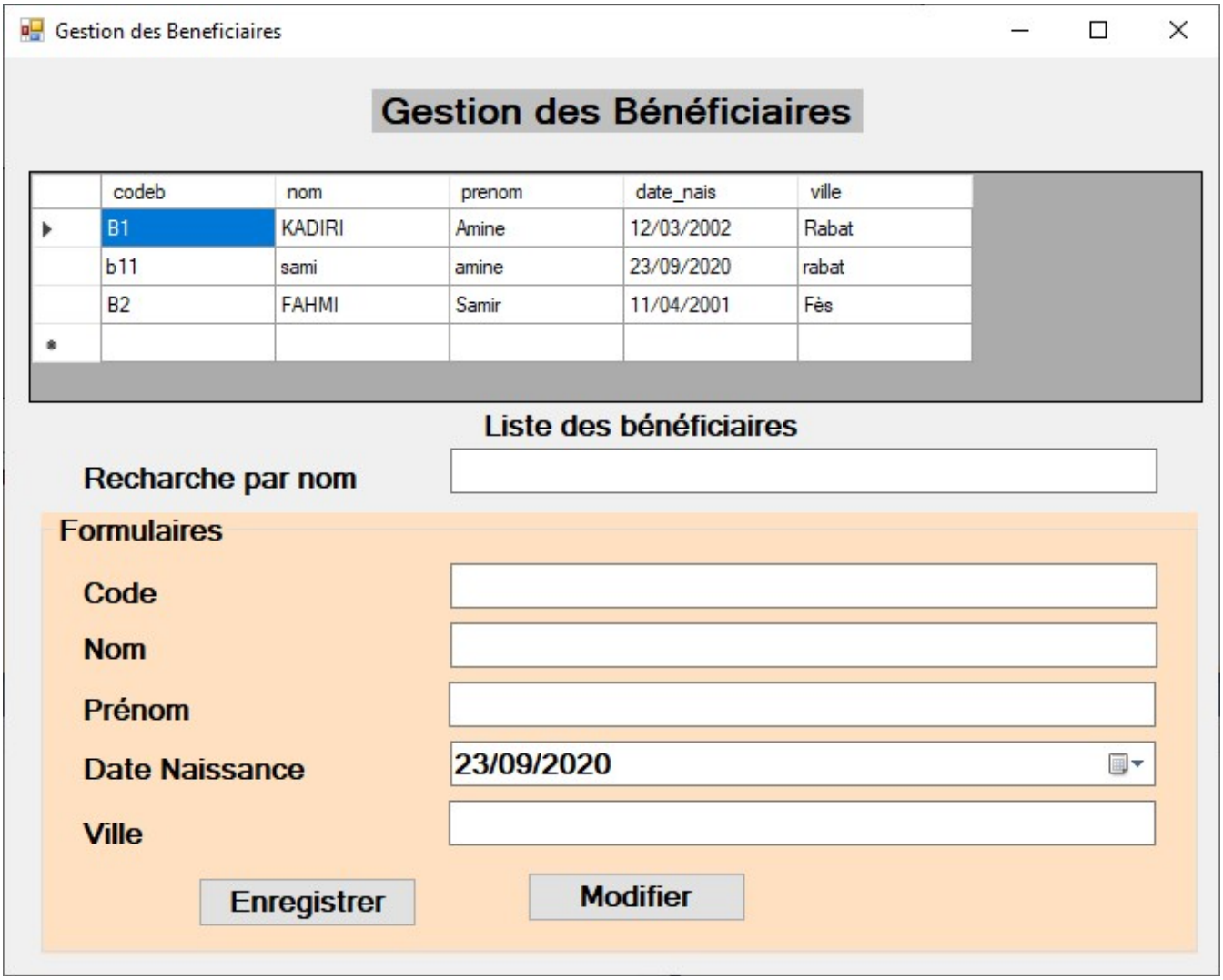
```
Private Sub updateBen(ByVal c As String, ByVal n As String, ByVal p As String, ByVal
dn As Date, ByVal v As String)
...
End Sub
```

Les étapes à suivre :

- Vérifier la validité du formulaire ;
- Vérifier l'existence du code bénéficiaire ;
- Envoyer la requête de modification d'un nouveau bénéficiaire.

Interface de gestion des bénéficiaires avec une recherche instantanée

On ajoute à l'interface graphique précédente une zone texte de recherche instantanée selon les noms des bénéficiaires : **TextBox** de recherche a comme **Name= txtSearch**



➢ Procédure de recherche par nom

```
Private Sub searchBen(ByVal n As String)
...
End Sub
```

Les étapes à suivre :

Méthode 1 :

- ❖ Préparer une requête de sélection des bénéficiaires ayant le nom qui ressemble à l'argument **n** ;
- ❖ Envoyer la requête puis charger un **DataTable** ;

- Afficher le résultat sur **DataGridView**.

Méthode 2 :

- Préparer une requête pour lire tous les bénéficiaires ;
- Envoyer la requête puis charger un **DataTable** ;
- Préparer un objet **DataView**
- Préciser le filtre de cet objet via la propriété **RowFilter**
- Afficher le résultat sur **DataGridView**.

➢ Procédure événementielle de **TextBox txtSearch**

```
Private Sub txtSearch_TextChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles txtSearch.TextChanged

    searchBen(txtSearch.Text)

End Sub
```