Paginer les données côté serveur, mettre en cache côté client

Vous voulez sélectionner des lignes dans une table, mais celle-ci comporte trop de lignes pour qu'il soit réaliste de les ramener en une seule fois sur le client. Vous allez alors « paginer » votre requête, organiser celle-ci en lots successifs qui vont récupérer les lignes voulues depuis le serveur. Quand vous aurez chargé une page de données, vous voudrez peut-être la conserver localement pour pouvoir y revenir sans avoir à le recharger depuis le serveur. Vous allez alors « mettre en cache » ces pages sur le client.

Dans cet exemple, nous travaillerons sur la table **Sales.SalesOrderDetails** de la base de données **AdventureWorks2008**.

Commençons par une requête qui nous permette de préciser la taille de la page de données (c'est-à-dire le nombre de lignes par page), et le numéro de la page voulue.

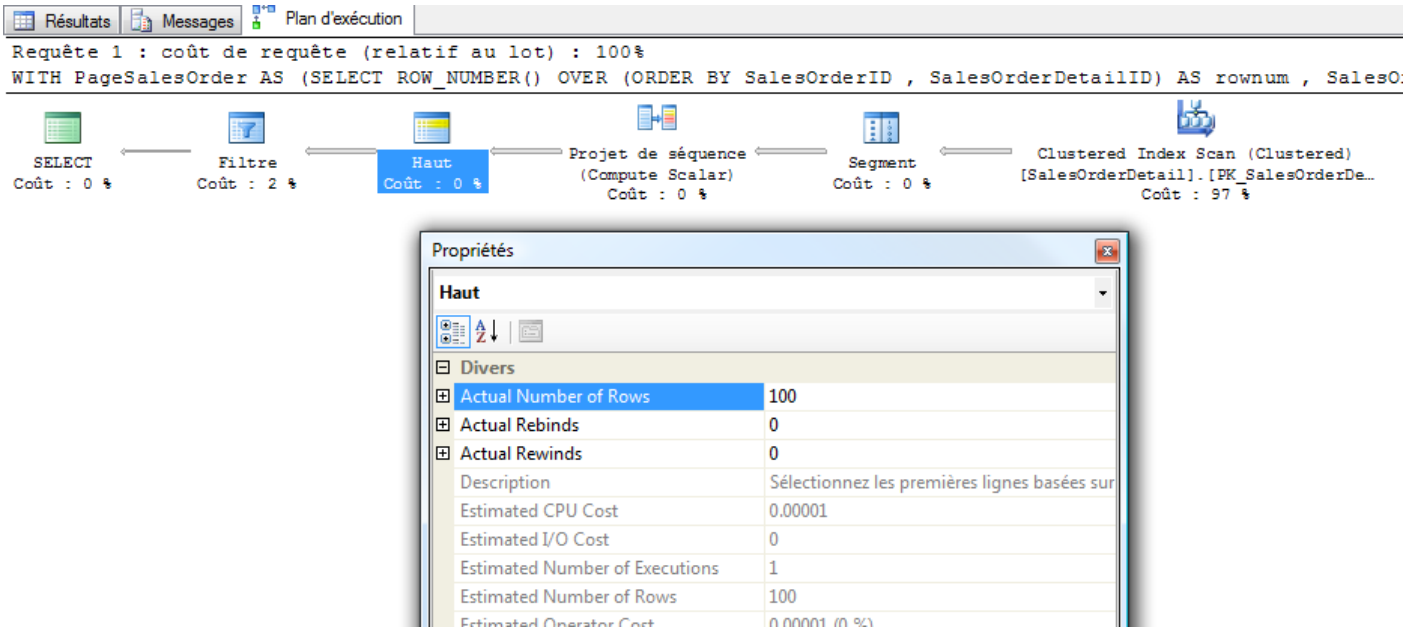
```
-- Demandons la deuxième page de SalesOrderDetail,
-- chaque page est basée (ordonnée) sur SalesOrderID, SalesOrderDetailID
-- et comporte 50 lignes

DECLARE @pagesize AS INT ,
        @pagenum AS  INT;

SET @pagesize = 50;
SET @pagenum  = 2;

WITH PageSalesOrder AS
          (SELECT           ROW_NUMBER() OVER (ORDER BY SalesOrderID , SalesOrderDetailID) AS rownum ,
                            SalesOrderID                                                              ,
                            SalesOrderDetailID                                                        ,
                            ProductID                                                                 ,
                            OrderQty                                                                  ,
                            UnitPrice                                                                 ,
                            UnitPriceDiscount
                   FROM     AdventureWorks2008.Sales.SalesOrderDetail
          )
 SELECT             rownum             ,
                    SalesOrderID       ,
                    SalesOrderDetailID ,
                    ProductID          ,
                    OrderQty           ,
                    UnitPrice          ,
                    UnitPriceDiscount
          FROM      PageSalesOrder
          WHERE     rownum  > @pagesize * (@pagenum-1)
          AND       rownum <= @pagesize * @pagenum
          ORDER BY rownum;
```

Que fait cette requête ? comment est-elle construite ?

Pour l'essentiel, elle repose sur une CTE (Common Table Expression), qui va créer dans le jeu résultant une colonne de type Entier, dont le contenu va être le numéro de la ligne de l'extraction basée sur les colonnes choisies. C'est ce que fait le ROW_NUMBER( ) OVER (ORDER BY…

La requête extrait ensuite du jeu ramené par la CTE les lignes demandées. On pourrait penser que la CTE ramène toutes les lignes, puisque rien dans son corps ne semble donner de filtre. En réalité, si on regarde le plan d'exécution

de cette requête, et en particulier l'opérateur « Haut » qui indique le nombre de lignes ramenées par la CTE, on voit que cette CTE ne ramène que 100 lignes, c'est-à-dire les 2 pages de 50 lignes.



La requête ne demande pas plus de lignes que ce qui est déductible du WHERE ! c'est toute la puissance des CTE qui est ici mise en jeu…

Cette approche (pagination à la demande), est adaptée à une navigation principalement vers l'avant (on demande une page, puis sa suivante, etc..) sur de tables de volume moyen (moins de 5 millions de lignes). Après chaque requête, les pages demandées sont mises en cache sur le serveur, ce qui permettra au scan demandé par la requête suivante d'être fait en mémoire, seules les nouvelles lignes demandées nécessiteront un accès physique.

Il ne nous reste plus qu'à « empaqueter » cette requête sur le serveur. Nous disposons de 2 solutions : une procédure stockée, ou bien (puisqu'il s'agit d'une seule ligne de requête), une fonction table incluse.

Les deux techniques retourneront le résultat sous forme d'un jeu d'enregistrements.

Voici le code de création de la procédure stockée :

```
CREATE PROCEDURE [dbo].[PaginationAdHoc]
       @pageSize int = 10,
       @pageNum int = 1
AS
BEGIN
       SET NOCOUNT ON;

WITH PageSalesOrder AS
          (SELECT          ROW_NUMBER() OVER (ORDER BY SalesOrderID , SalesOrderDetailID) AS rownum ,
                           SalesOrderID                                                              ,
                           SalesOrderDetailID                                                        ,
                           ProductID                                                                 ,
                           OrderQty                                                                  ,
                           UnitPrice                                                                 ,
                           UnitPriceDiscount
                  FROM     AdventureWorks2008.Sales.SalesOrderDetail
          )
 SELECT            rownum            ,
                   SalesOrderID      ,
                   SalesOrderDetailID ,
                   ProductID         ,
                   OrderQty          ,
                   UnitPrice         ,
```

```
                UnitPriceDiscount
        FROM    PageSalesOrder
        WHERE   rownum  > @pagesize * (@pagenum-1)
        AND     rownum <= @pagesize * @pagenum
        ORDER BY rownum;
END
```

Cette procédure stockée attend les 2 paramètres nécessaires, mais des valeurs par défaut leur sont attribuées (nous verrons plus loin comment les utiliser).

Voici un exemple d'utilisation de cette procédure stockée, qui demande la 50ème page de 100 lignes :

```
EXECUTE [tests].[dbo].[PaginationAdHoc]
                    @pageSize = 100 ,
                    @pageNum = 50
```

Remarquez l'utilisation directe des paramètres demandés, sans déclaration préalable, en leur affectant leur valeur. Cette syntaxe permet de passer les paramètres dans l'ordre qui nous convient, mais les noms des paramètres doivent être exactement ceux qui sont déclarés dans le corps de la procédure stockée.

Pour la fonction table incluse, il nous faut supprimer la clause ORDER BY de cette requête, celle-ci étant interdite dans les fonctions tables incluses, qui sont quasiment des vues.

Voici le code de création de la fonction :

```
CREATE FUNCTION [dbo].[GetPageFromSalesOrderDetails]
(
    @pageNum int,
    @pageSize int
)
RETURNS TABLE
AS
RETURN
(
    WITH PageSalesOrder AS
            (SELECT        ROW_NUMBER() OVER (ORDER BY SalesOrderID , SalesOrderDetailID) AS rownum ,
                           SalesOrderID                                                              ,
                           SalesOrderDetailID                                                        ,
                           ProductID                                                                 ,
                           OrderQty                                                                  ,
                           UnitPrice                                                                 ,
                           UnitPriceDiscount
                 FROM     AdventureWorks2008.Sales.SalesOrderDetail
            )
     SELECT            rownum            ,
                SalesOrderID       ,
                SalesOrderDetailID ,
                ProductID          ,
                OrderQty           ,
                UnitPrice          ,
                UnitPriceDiscount
            FROM     PageSalesOrder
            WHERE    rownum  > @pageSize * (@pageNum-1)
            AND      rownum <= @pageSize * @pageNum
)
```

Nous devrons mettre ce ORDER BY dans la requête appelante, dont voici un exemple qui demande la 50ème page de 100 lignes :

```
SELECT            rownum            ,
                  SalesOrderID      ,
                  SalesOrderDetailID ,
                  ProductID         ,
                  OrderQty          ,
                  UnitPrice         ,
                  UnitPriceDiscount
         FROM     [tests].[dbo].[GetPageFromSalesOrderDetails] (50 , 100)
         ORDER BY rownum
```

Comment allons-nous utiliser cette procédure stockée et cette fonction depuis VFP, et depuis un Business Object de StrataFrame ?

Pour VFP, je vous propose de créer un CursorAdapter pour accéder à ces données. J'ai ici simplement repris le code généré par l'Explorateur de données.

```
LOCAL oCA as CursorAdapter
LOCAL oConn as ADODB.Connection
LOCAL oRS as ADODB.Recordset
LOCAL oCmd as ADODB.Command
LOCAL oException AS Exception
LOCAL cConnString

* Gestion de la Connexion - Insérez le code de la connexion
cConnString = [Provider=SQLOLEDB.1;Data Source=acervistalevy\sql2008;Initial Catalog=tests;User
ID=;Password=;Integrated Security=SSPI]

TRY
    oConn  = createobject("ADODB.Connection")

    * Vérifez que vous disposez des identifiants de connexion (utilisateur et mot de passe)
    * s'ils ne sont pas spécifiés dans la chaîne de connexion.
    *   ex. oConn.Open(cConnString, userid, password)
    oConn.Open(cConnString)

    oRS = CREATEOBJECT("ADODB.Recordset")
    oRS.ActiveConnection = oConn

    oCmd=CREATEOBJECT("ADODB.Command")
    oCmd.ActiveConnection=oConn

    oCA=CREATEOBJECT("CursorAdapter")
    oCA.DataSourceType = "ADO"
    oCA.DataSource = oRS
    oCA.MapBinary = .T.
    oCA.MapVarchar = .T.
    oCA.AddProperty("PageSize",100)
    oCA.AddProperty("PageNum",50)

    oCA.Alias = [PageDeSalesOrderDetail]
```

Pour plus de modularité, je crée les 2 propriétés qui stockeront les valeurs des paramètres à passer à la procédure stockée ou à la fonction. Seul le SelectCmd du CursorAdapter va être différent, selon qu'on utilise la procédure stockée ou la fonction.

Utilisation de la procédure stockée

```
    oCA.SelectCmd="EXEC [PaginationAdHoc] @pageSize = ?oCA.pageSize, @pageNum = ?oCA.pageNum"
```

Utilisation de la fonction

```
oCA.SelectCmd=;
    "SELECT "+;
                  "rownum       ,"+;
              "SalesOrderID     ,"+;
              "SalesOrderDetailID ,"+;
              "ProductID        ,"+;
              "OrderQty         ,"+;
              "UnitPrice        ,"+;
              "UnitPriceDiscount "+;
     "FROM     [tests].[dbo].[GetPageFromSalesOrderDetails] (?oCA.PageNum, ?oCA.PageSize) "+;
     "ORDER BY rownum"
```

Dans les deux utilisations, remarquez bien le passage de paramètre avec le point d'interrogation précédent le nom de la variable : **`?oCA.PageNum`** et **`?oCA.PageSize`**.
C'est la seule façon de se protéger contre une éventuelle injection SQL.

Et dans StrataFrame ?

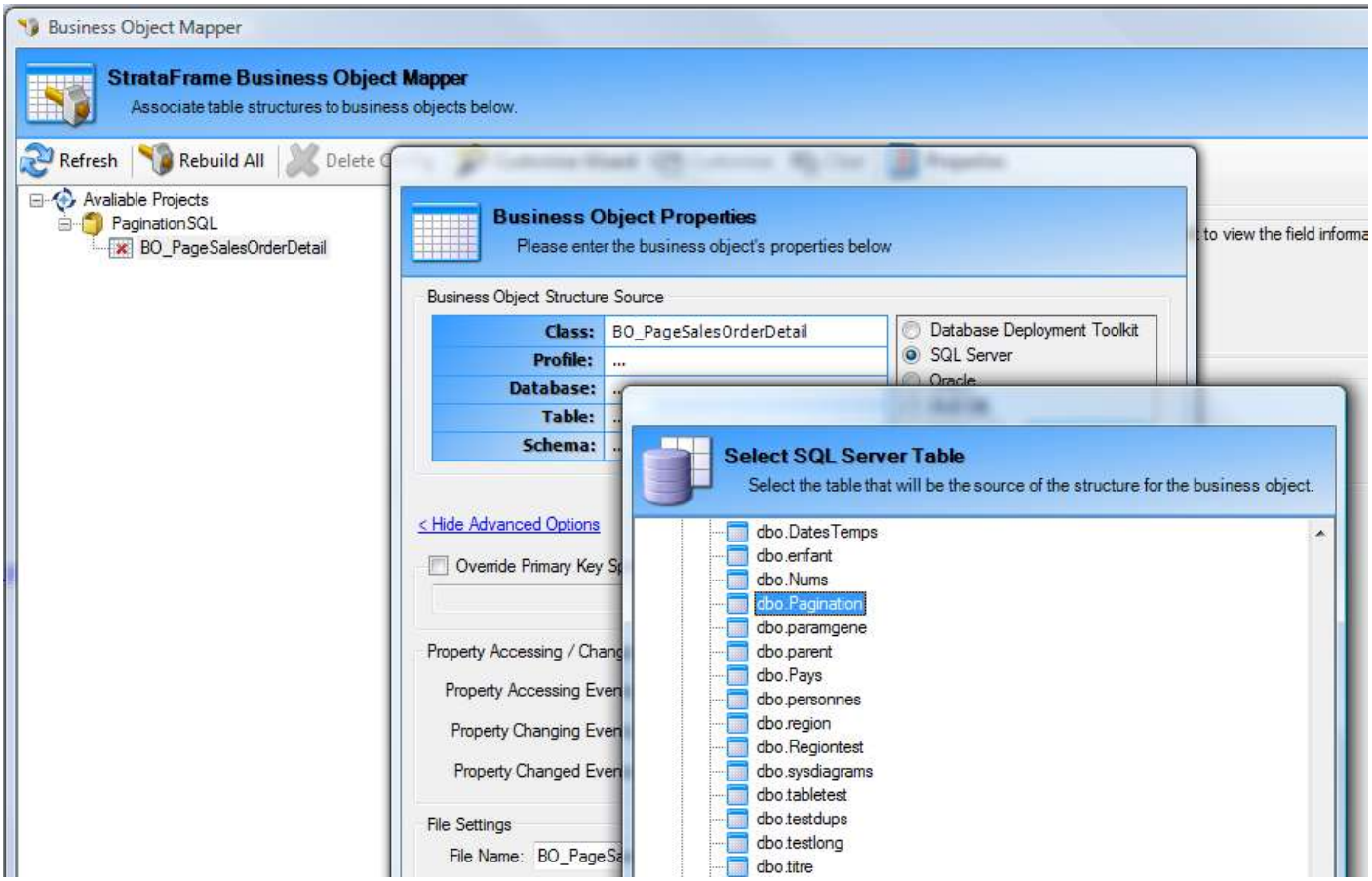
Il nous faut d'abord créer le Business Object, et le mapper sur les champs qui seront générés par la procédure stockée ou par la fonction. La procédure stockée ainsi que la fonction créent un champ qui n'existe pas dans la table sous-jacente (le RowNum), et ne ramènent pas tous les champs de cette table. Nous allons donc créer une vue, dont l'usage sera limité à l'établissement du mappage ; une fois celui-ci terminé, nous pourrons la supprimer.

Voici le code de la vue :

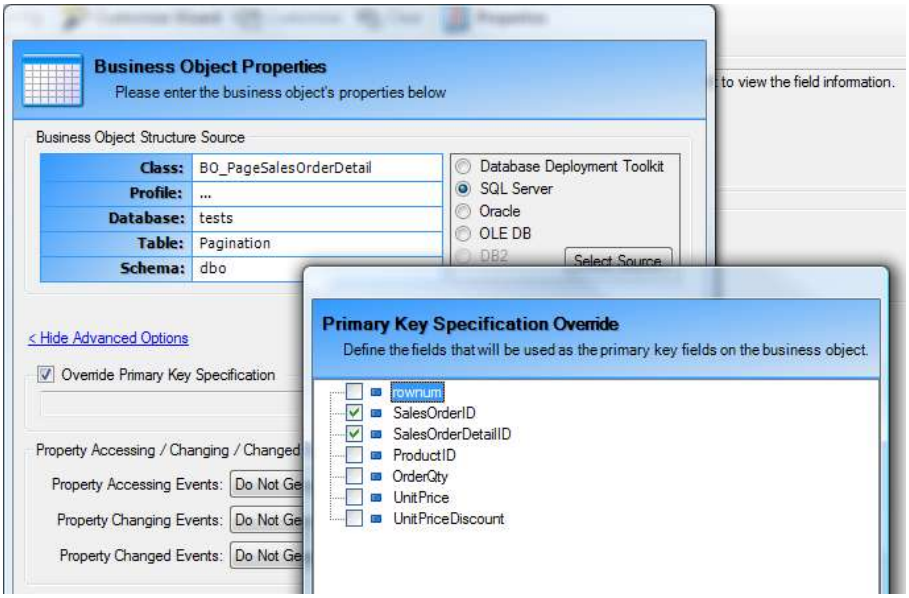
```
CREATE VIEW Pagination AS
SELECT        rownum             ,
              SalesOrderID       ,
              SalesOrderDetailID ,
              ProductID          ,
              OrderQty           ,
              UnitPrice          ,
              UnitPriceDiscount
       FROM   dbo.GetPageFromSalesOrderDetails (DEFAULT , DEFAULT)
```

Pour cette vue, nous utilisons la fonction créée précédemment, en lui passant les paramètres par défaut.

Nous pouvons alors effectuer le mappage : la vue que nous venons de créer est visible comme une source de données, de la même façon qu'une table.



N'oublions pas de forcer la définition de la clé primaire sur le mappage, puisqu'il s'agit d'une vue non indexée : ceci permettra au générateur de mappage d'implémenter correctement toutes les propriétés qui seront éventuellement utilisées (dont la propriété **`PrimaryKeyField`**). Je reprends ici les champs utilisés comme PK dans la table sous-jacente à la vue.



(on pourrait supprimer maintenant la vue créée sur le serveur)

Notre structure est prête, il nous faut maintenant la remplir avec les données que nous allons chercher sur le serveur. À partir de ce point, je vous propose d'utiliser la procédure stockée (on aurait pu tout aussi bien utiliser la fonction table en ligne).

Nous avons 2 possibilités : utiliser une méthode **`FillByStoredProcedure`**, ou une méthode **`FillDataTable`**.

Utilisation de **`FillByStoredProcedure`**

```
Public Sub PopulateByStoredProcedure(ByVal PageNum As Integer, ByVal PageSize As Integer)
    Dim loNum As SqlParameter
    Dim loPage As SqlParameter
    loNum = New SqlParameter("@PageNum", Data.SqlDbType.Int)
    loPage = New SqlParameter("@PageSize", Data.SqlDbType.Int)

    loNum.Value = PageNum
    loPage.Value = PageSize

    Me.FillByStoredProcedure("dbo.PaginationAdHoc", loPage, loNum)

End Sub
```

Utilisation de **`FillDataTable`**

```
Public Sub PopulateByFillDataTable(ByVal PageNum As Integer, ByVal PageSize As Integer)
    Dim loSQL As New SqlCommand
    loSQL.Parameters.AddWithValue("@PageNum",PageNum)
    loSQL.Parameters.AddWithValue("@PageSize",PageSize)

    loSQL.CommandText="EXEC dbo.PaginationAdHoc @PageSize, @PageNum"

    Me.FillDataTable(loSQL)
End Sub
```

Le code exécuté sur le serveur est le même (vous pouvez le vérifier en lançant une trace au Profiler SQL), le résultat est identique.

Chaque fois que l'une ou l'autre de ces procédure est appelée, le contenu du BO est remplacé par la nouvelle page de données ; comment pourrions-nous mettre en cache cette nouvelle page, pour ne pas avoir à la rappeler depuis le serveur si nous en avons encore besoin ? comment pourrions-nous ajouter la nouvelle page aux données déjà récupérées, si nous ne l'avons pas déjà fait ?

Commençons par créer des fonctions Get identiques à nos méthodes Fill déjà créées : les fonctions Get fonctionnent comme des méthodes Fill, mais au lieu de peupler le Business Object, elles renvoient une dataTable.

```
Public Function GetPageByStoredProcedure _
    (ByVal PageNum As Integer, ByVal PageSize As Integer) _
    As DataTable
    Dim loNum As SqlParameter
    Dim loPage As SqlParameter
    loNum = New SqlParameter("@PageNum", Data.SqlDbType.Int)
    loPage = New SqlParameter("@PageSize", Data.SqlDbType.Int)

    loNum.Value = PageNum
    loPage.Value = PageSize

    Return Me.GetByStoredProcedure("dbo.PaginationAdHoc", loPage, loNum)
End Function

Public Function GetPageByGetDataTable _
    (ByVal PageNum As Integer, ByVal PageSize As Integer) _
    As DataTable
    Dim loSQL As New SqlCommand
    loSQL.Parameters.AddWithValue("@PageNum", PageNum)
    loSQL.Parameters.AddWithValue("@PageSize", PageSize)

    loSQL.CommandText = "EXEC dbo.PaginationAdHoc @PageSize, @PageNum"
```

```
        Return Me.GetDataTable(loSQL)
    End Function
```

Pour la suite de notre exemple, nous utiliserons la méthode **PopulateByStoredProcedure**, et la fonction **GetPageByStoredProcedure**.

Nous aurons besoin de façon persistante de la taille des pages, du numéro de la page à obtenir ou en cours, de la liste des pages déjà mises en cache ; il nous faudra aussi savoir si nous voulons écraser les données ou les ajouter, et si nous voulons conserver en cache les pages de données.

Créons donc dans notre Business Object les propriétés nécessaires.

```
#Region "Propriétés de Pagination"

    Private PagesPresentes As ArrayList = New ArrayList()

    Private _PageSize As Integer = 1000
    Private _PageNum As Integer = 1
    Private _CachePages As Boolean = False
    Private _AdditiveMode As Boolean = True

    <Category("Pagination"), _
    Description("Taille de la page de données"), _
    Browsable(True)> _
    Public Property PageSize As Integer
        Get
            Return _PageSize
        End Get
        Set(ByVal value As Integer)
            _PageSize = value
        End Set
    End Property

    <Category("Pagination"), _
    Description("Numéro de la page"),
    Browsable(True)>
    Public Property PageNum As Integer
        Get
            Return _PageNum
        End Get
        Set(ByVal value As Integer)
            _PageNum = value
        End Set
    End Property

    <Category("Pagination"), _
    Description( _
        "En mode Additive, les pages sont ajoutées au BO," _
        &" en mode NON additive, chaque page écrase le contenu du BO"), _
    Browsable(True)>
    Public Property AdditiveMode As Boolean
    Get
            Return _AdditiveMode
    End Get
    Set(ByVal value As Boolean)
            _AdditiveMode=value
    End Set
    End Property

    <Category("Pagination"), _
    Description("Mise en cache local des pages récupérées"), _
    browsable(True)>
    Public Property CachePages As Boolean
    Get
            Return _CachePages
```

```
    End Get
    Set(ByVal value As Boolean)
            _CachePages=value
    End Set
    End Property

#End Region
```

Il ne nous reste plus qu'à créer les méthode **FillNextPage** et **FillPreviousPage**, en prenant en compte nos options possibles.

Voici le code de la méthode **FillNextPage** :

```
    ''' <summary>
    ''' Remplit le BO avec la page suivante:
    ''' En mode Additive, si la page n'a pas déjà été chargée,
    ''' on la charge depuis le serveur et on l'ajoute à celle en cours
    '''
    ''' en mode NON additive, si le CachePages est True et que la page a déja été chargée,
    ''' on la remonte depuis le cache
    ''' sinon on la recharge depuis le serveur
    ''' </summary>
    ''' <param name="Additive"></param>
    ''' <remarks>le cache est implémenté par des snapshots nommés</remarks>
    Public Sub FillNextPage(Optional ByVal Additive As Boolean = False)

        If Additive
            ' On charge la page uniquement si elle n'a pas été déjà chargée
            ' (donc présente dans le tableau des PagesPresentes)
            if Not Me.PagesPresentes.Contains(Me.PageNum+1) Then
                Dim loPage As DataTable = Me.GetPageByStoredProcedure(Me.PageNum + 1, Me.PageSize)
                Me.CopyDataFrom(loPage, BusinessCloneDataType.AppendDataToTableFromCompleteTable)
            End If
        Else
            If Me.CachePages=True
                'si la page est déjà dans le cache, on la recharge depuis le cache
                'sinon, on Populate et on la met en cache
                Dim SnapPageNum As Integer=Me.PageNum+1
                Dim SnapPage As String="Snap"+SnapPageNum.ToString

                If Me.PagesPresentes.Contains(SnapPageNum)
                    Me.RestoreCurrentDataTableSnapshot(SnapPage,true)
                Else
                    ' si la SnapPage demandé n'existe pas, on doit la charger, et en faire un snapshot
                    Me.PopulateByStoredProcedure(SnapPageNum, Me.PageSize)
                    Me.SaveCurrentDataTableToSnapshot(SnapPage)
                End If

            else
                Me.PopulateByStoredProcedure(Me.PageNum + 1, Me.PageSize)
            End If

        End If

        Me.PageNum += 1
        Me.PagesPresentes.Add(Me.PageNum)
    End
```

En mode « Additive », on veut rajouter la page suivante après les données présentes dans la DataTable du BO, uniquement si on ne l'a pas déjà fait. On commence donc par vérifier que cette page n'est pas présente dans le tableau (ArrayList) « PagesPresentes ». Si cette page n'y figure pas, on la crée en appelant la fonction **GetPageByStoredProcedure**, et on utilise simplement la méthode **CopyDataFrom** présentée par StrataFrame pour l'ajouter après les données actuelles.

En mode non Additive, si on ne veut pas mettre les pages en cache (la propriété CachePages est à False), il nous suffit d'appeler la méthode **PopulateByStoredProcedure** ; les données de la page suivante remplacent alors la page en cours. Si on a choisi la mise en cache, on commence par vérifier si cette page n'est pas déjà présente (comme ci-dessus). Si cette page n'existe pas, nous la chargeons dans le BO par la méthode **PopulateByStoredProcedure**, et nous la mettons en cache en utilisant la méthode **SaveCurrentDataTableToSnapshot** présentée par StrataFrame. Si la page est déjà en cache, il nous suffit de la restaurer par la méthode **RestoreCurrentDataTableSnapshot**. Ces deux méthode permettent de nommer les snapshot conservés par une clé de type String.

La méthode **FillPreviousPage** est quasiment identique :

```
''' <summary>
''' Remplit le BO avec la page suivante:
''' En mode Additive, si la page n'a pas déjà été chargée,
''' on la charge depuis le serveur et on l'ajoute à celle en cours
''' par un Merge pour mettre la nouvelle page en tête de l'existant
'''
''' en mode NON additive, si le CachePages est True et que la page a déja été chargée,
''' on la remonte depuis le cache
''' sinon on la recharge depuis le serveur
''' </summary>
''' <param name="Additive"></param>
''' <remarks></remarks>
'''
Public Sub FillPreviousPage(Optional ByVal Additive As Boolean = False)
    If Me.PageNum > 1 Then
        If Additive Then
            ' On charge la page uniquement si elle n'a pas été déjà chargée
            ' (donc présente dans le tableau des PagesPresentes)
            if Not Me.PagesPresentes.Contains(Me.PageNum-1) Then
                Dim loPage As DataTable = Me.GetPageByStoredProcedure(Me.PageNum - 1, Me.PageSize)
                loPage.Merge(Me.CurrentDataTable)
                Me.CopyDataFrom(loPage, BusinessCloneDataType.ClearAndFillFromCompleteTable)
            End If
        Else
            If Me.CachePages=True
            'si la page est déjà dans le cache, on la recharge depuis le cache
            'sinon, on Populate et on la met en cache
            Dim SnapPageNum As Integer=Me.PageNum-1
            Dim SnapPage As String="Snap"+SnapPageNum.ToString

            If Me.PagesPresentes.Contains(SnapPageNum)
                Me.RestoreCurrentDataTableSnapshot(SnapPage,true)
            Else
                ' si la SnapPage demandé n'existe pas, on doit la charger, et en faire un snapshot
                Me.PopulateByStoredProcedure(SnapPageNum, Me.PageSize)
                Me.SaveCurrentDataTableToSnapshot(SnapPage)
            End If

            Else
                Me.PopulateByStoredProcedure(Me.PageNum - 1, Me.PageSize)
            End If

        End If
        Me.PageNum -= 1
    End If
End
```

La seule différence est en mode Additive : il faut mettre la nouvelle page avant les données en cours. On va utiliser la métode native .Net Merge pour fusionner la page en cours (la **CurrentDataTable**) après la nouvelle page, et copier cet ensemble dans la DataTable du BO, avec la méthode StrataFrame **CopyDataFrom** en lui passant le paramètre **ClearAndFillFromCompleteTable**.

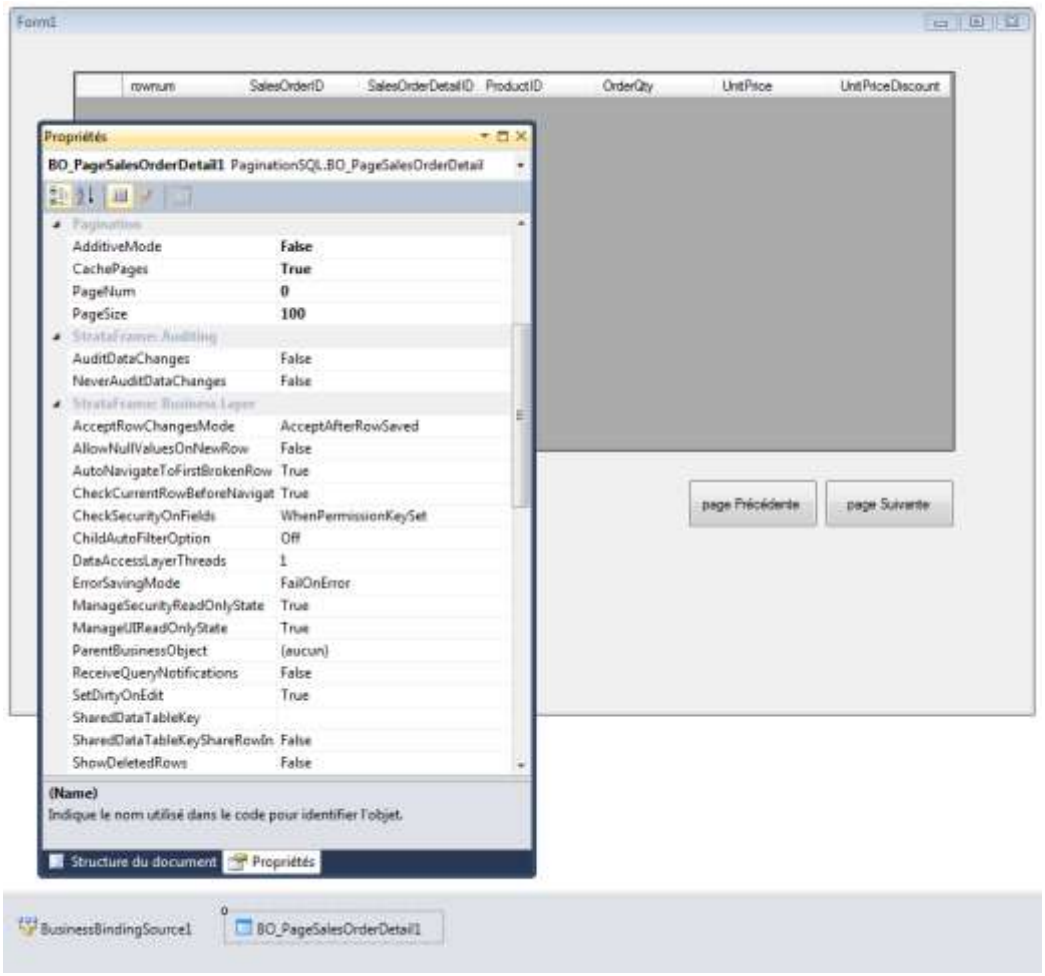
Pour tester ce Business Object, créez un form dérivé de la classe StandardForm de StrataFrame, jetez-y ce BO, ajoutez un DataGridView natif de .Net, un BusinessBindingSource de StrataFrame pour lier ce grid au BO. Définissez la valeur de PageNum à 0.

Un double click sur le BO en design du form nous amène sur la méthode ParentFormLoading que nous implémentons simplement comme suit :

```
Private Sub BO_PageSalesOrderDetail1_ParentFormLoading() _
    Handles BO_PageSalesOrderDetail1.ParentFormLoading
        Me.BO_PageSalesOrderDetail1.FillNextPage()
End Sub
```

Puis nous ajoutons les boutons cmdPageSuivante et cmdPagePrecedente, dont nous implémentons les méthodes click ainsi :

```
Private Sub cmdPageSuivante_Click( ByVal sender As System.Object,  ByVal e As System.EventArgs) _
    Handles cmdPageSuivante.Click
        Me.BO_PageSalesOrderDetail1.FillNextPage(Me.BO_PageSalesOrderDetail1.AdditiveMode)
End Sub

Private Sub cmdPagePrecedente_Click( ByVal sender As System.Object,  ByVal e As System.EventArgs) _
    Handles cmdPagePrecedente.Click
        Me.BO_PageSalesOrderDetail1.FillPreviousPage(Me.BO_PageSalesOrderDetail1.AdditiveMode)
End Sub
```



Modifiez les valeurs de la taille de la page, passez en mode Additive, avec et sans le cache…