Stocker et Annuler les mises à jour dans un formulaire

par Charles A.

Date de publication : 31/01/2006

Dernière mise à jour : 31/01/2006

Thèmes abordés : . identification des événements liés aux mises à jour . codage d'une solution de stockage . codage d'une solution d'annulation . implémentation dans un formulaire Niveau requis : moyen / avancé

I - Introduction

Le but de ce tutoriel est de proposer une solution de stockage et d'annulation de mise à jour de données dans Access.

Sur le forum, se trouvent de nombreux sujets abordant les problématiques suivantes :

. comment savoir qui a modifié ma table ?

. comment connaître la dernière mise à jour de données ?

. comment annuler les mises à jour d'un utilisateur ?

. peut-on déclencher un code lors de la modification de données ?

Ces questions récurrentes méritent qu'on s'y penche, et qu'on tente de leur apporter une réponse.

Ce champ de questions est issu d'une même problématique : **il n'y a pas dans Access et son moteur JET de *triggers***.

Qu'est-ce qu'un **trigger** ?

Déclencheur (trigger)

Un déclencheur est une procédure stockée qui s'exécute lors d'une tentative d'action particulière (insertion, modification, suppression) sur une table ou sur une vue.

Plus d'informations sur ce site Les Triggers en SQL.

Dans MS Access, les modifications / insertions / suppressions dans une table / requête n'occasionnent aucun événement et ne peuvent actionner de déclencheur (trigger), notre dernier recours est d'utiliser les **formulaires** qui sont aptes à manipuler ce type d'événements.

II - Méthodologie - Evénements

L'objectif est de pouvoir stocker les mises à jour et de permettre leur annulation.

Le tutoriel est prévu pour fonctionner sur un ***formulaire lié à une source de données (dépendant)****.*

Dans le cas de formulaire indépendant, il suffirait de coder un traçage sur la sub de mise à jour.

Pour annuler une mise à jour il nous faut :

. cas d'une suppression : stocker les valeurs de l'ancien enregistrement ;

. cas d'une modification : stocker les anciennes valeurs qui ont été écrasées ; . cas d'une insertion : stocker la clé du nouvel enregistrement en vue de son éventuelle suppression ;

. tous cas : enregistrer le login de la personne qui modifie, et l'heure de mise à jour.

Pour parvenir au but recherché, c'est à dire tracer toutes les mises à jour de données et permettre de les annuler, nous allons travailler sur les formulaires.

Ce tutoriel ne pourra pas fonctionner dans les cas suivants :

. saisies directes dans des tables ou requêtes ;

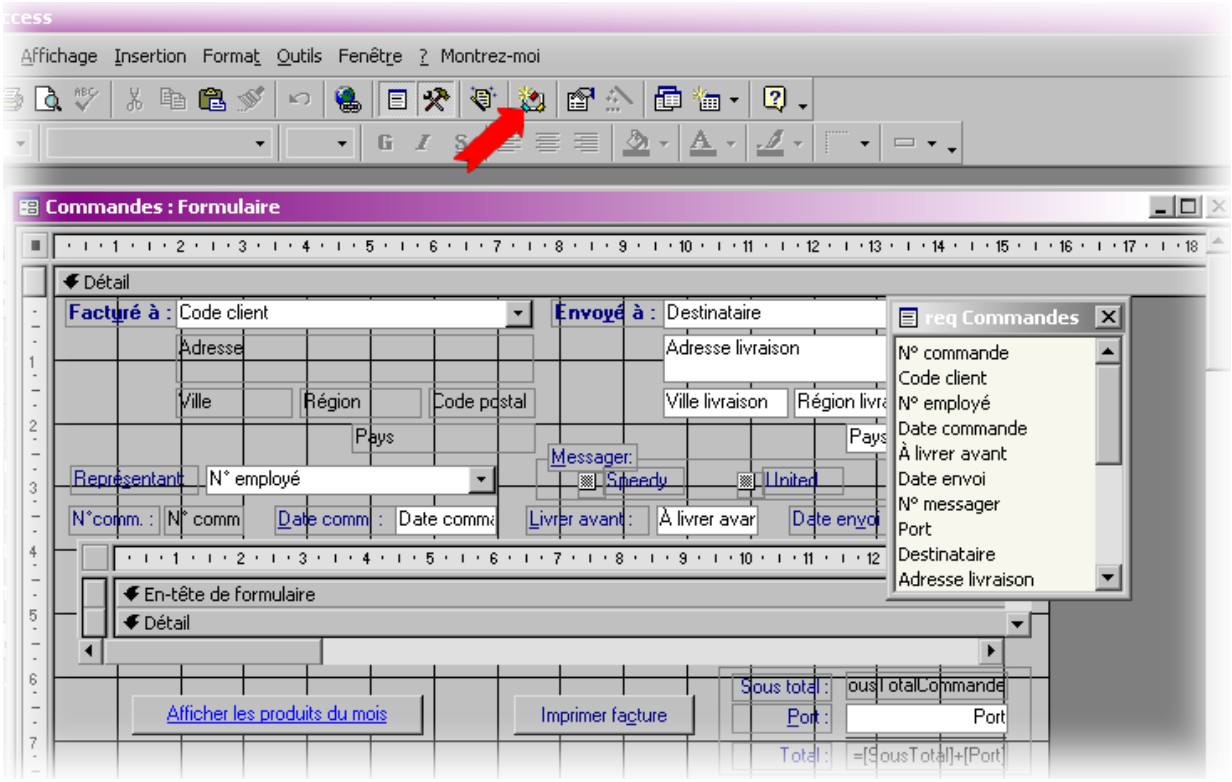
. imports de données externes ;

. requête ajout ou mise à jour.

Pour stocker les mises à jour, nous devons déterminer quels événements activent ces modifications.

Nous travaillons sur la base **comptoir.mdb** fournie dans le répertoire *Samples* de Microsoft Office.

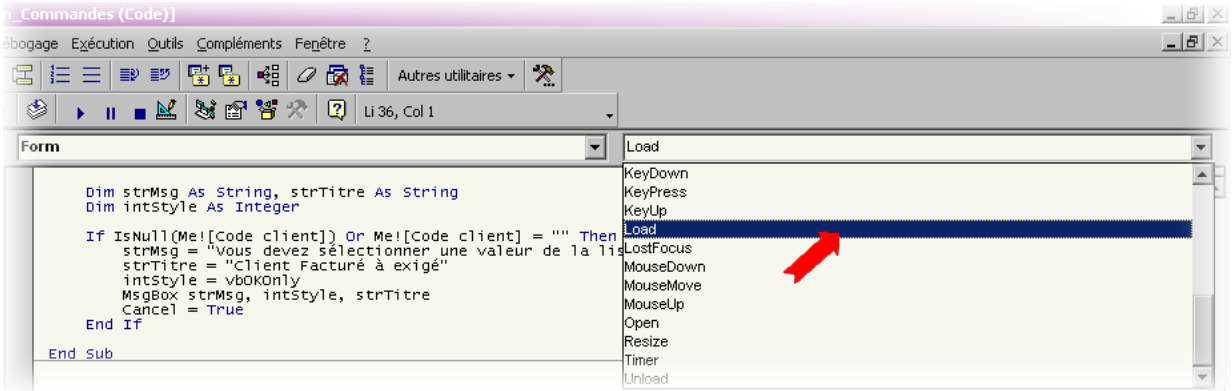
Nous ouvrons un formulaire en mode création, et nous explorons le code VBA.



Exploration du code VBA d'un formulaire

Dans les deux listes déroulantes nous sélectionnons **Form** (désignant le formulaire) dans celle de gauche qui représente la liste des objets du formulaire.

Et nous parcourons celle de droite qui désigne les méthodes et événements.



Parcours des méthodes et événements

Nous cherchons quels sont les événements les plus appropriés pour tracer les modifications.

. **traçage d'une insertion** : nous choisissons **AfterInsert**.

Seuls deux événements concernent un *INSERT*, Before et After. Si nous choisissons Before (avant) nous n'aurons pas accès aux nouvelles valeurs insérées, donc nous ne pourrons rien stocker.

Notre choix se porte sur **AfterInsert**, événement avec lequel nous connaîtrons les valeurs du nouvel enregistrement.

. **traçage d'une modification** : nous choisissons **BeforeUpdate**.

Ici aussi, deux événements prennent en charge un *UPDATE*, Before et After.

Si nous sélectionnons After, le jeu d'enregistrement aura été changé, et nous n'aurons plus accès aux anciennes valeurs, ce qui rend impossible toute annulation.

Notre choix se porte sur **BeforeUpdate** qui nous permet de travailler avec les anciennes valeurs (propriété **.OldValue**).

. **traçage d'une suppression** : nous n'avons pas d'autre événement que **Delete**.

III - Méthodologie - Stockage

Nous avons sélectionné les événements pertinents liés à une mise à jour quelle qu'elle soit dans la partie précédente.

Il nous faut maintenant choisir une méthode de stockage qui nous permette l'éventuelle annulation.

La méthode retenue peut faire débat, et je n'ai pas la certitude qu'elle soit nécessairement la meilleure.

Regardons l'éventail des possibilités :

. **duplication des données**

Si nous copions les anciennes données il sera facile de revenir en arrière. L'inconvénient est que cette méthode peut se révéler lourde, et qu'il faudrait dupliquer toutes les tables à surveiller.

. **stockage champ par champ**

Nous créons un enregistrement par champ modifié. L'inconvénient est qu'on ne peut annuler des modifications champ par champ, sinon on pourrait alors faire apparaître des *Null* interdits et provoquer une erreur.

L'analyse de cette solution montre que nous devons impérativement stocker la mise à jour par enregistrement.

. **stockage d'enregistrement sous forme SQL**

Nous pourrions stocker la mise à jour sous forme de chaine d'annulation SQL.

ex : Nous modifions le champ *Nom* pour l'employé dont l'id est *12*

La chaîne d'annulation serait

Chaîne d'annulation SQL
```
UPDATE Employés
SET Nom = "Ancienne Valeur"
WHERE Id = 12
```

. **stockage sur le recordset**

Nous allons stocker enregistrement par enregistrement les modifications marginales sur le recordset source du formulaire.

. en cas de suppression : nous enregistrons les anciennes valeurs ;

. en cas d'ajout : nous n'enregistrons que les valeurs de clé du nouvel enregistrement, ce afin de le retrouver ou de le supprimer (annulation).

. en cas de modification : nous enregistrons les anciennes valeurs uniquement pour les changements.

Dans la pratique nous allons constituer une chaîne de mise à jour :

. un mot clé sur 6 caractères : UPDATE / DELETE / INSERT

. une liste de valeurs : la position ordinale du champ dans le recordset suivi du signe égal et de sa valeur.

Nous avons choisi cette dernière méthode, nous allons créer une chaîne d'annulation qui opèrera sur le recordset du formulaire.

tblLogUpdates : Table

id	author	type	revertAction	revertUnique	form
34	sysOP	UPDATE	UPDATE1="Davolio"	UNIQUE0=1	Employés
35	sysOP	UPDATE	UPDATE1="Davoliomenthe", 3="Représentant(e)"	UNIQUE0=1	Employés

Mode de stockage des modifications

Les modifications seront stockées dans une table, afin de pouvoir les consulter et de laisser le choix à l'utilisateur de les annuler.

IV - Table des modifications

Il s'agit de la table qui va recevoir les mises à jour faites sur le formulaire.

Nous avons besoin de recueillir les informations suivantes :

. **login** de la personne qui modifie [**author**] ;

. **date / heure** la mise à jour [**datUpdate**] ;

. **type** de la modification : suppression, édition, ajout [**type**] ;

. **nom du formulaire** sur lequel a eu lieu la modification [**form**] ;

. **chaîne d'annulation** qui va permettre de rétablir les anciennes valeurs [**revertAction**] ;

. **chaîne unique** c'est ce qui va permettre d'identifier de manière unique l'enregistrement modifié (*nous reviendrons sur cette notion plus bas*) [**revertUnique**] .

Voici la fonction qui nous permet de créer la table.

Procédure de création de la table

```
Private Sub CreateTableLogUpdates()
''' procédure de création de la table tblLogUpdates
DoCmd.RunSQL "CREATE TABLE tblLogUpdates (id COUNTER, author TEXT, type TEXT, " & _
                    "revertAction MEMO, revertUnique TEXT, form TEXT, Lot TEXT, " & _
                    "Field TEXT, datUpdate DATE, selection YESNO);"

End Sub
```

revertAction est notre chaîne d'annulation, son type est **MEMO** car il faut un important stockage dans le cas où un champ de type **Mémo** serait modifié, la limite du type **String** de 255 caractères serait trop vite atteinte.

Voici la fonction qui permet de recueillir le login de la personne qui effectue la mise à jour.

Fonction qui retourne le login de la session

```
'''""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""'''
```

Fonction qui retourne le login de la session

```
'
'                    API advapi32.dll
'
'''""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""'''
' récupérer le login Windows
Private Declare Function apiGetUserName Lib "advapi32.dll" Alias _
        "GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long

Private Function GetLogin() As String
' Retourne le nom d'usager fourni lors du branchement au réseau.
Dim lngLen As Long, lngX As Long
Dim strUserName As String
strUserName = String$(254, 0)
lngLen = 255
lngX = apiGetUserName(strUserName, lngLen)
If lngX <> 0 Then
    GetLogin = Left$(strUserName, lngLen - 1)
Else
    GetLogin = ""
End If

End Function
```

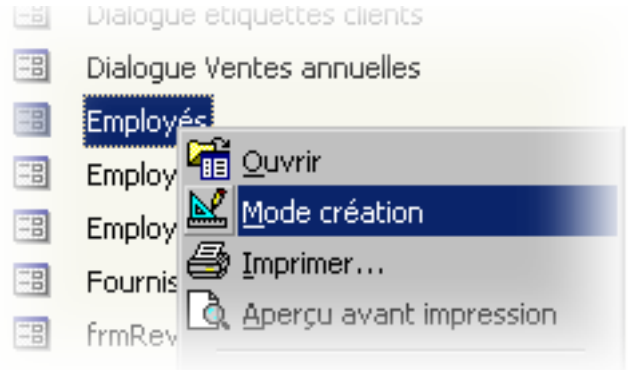
V - Mise en place de la solution

La solution se veut simple d'emploi, toutes les fonctions sont stockées dans un module ; **modLogUpdatesDAO**

Sur l'événement chargement du formulaire nous allons activer le traçage.

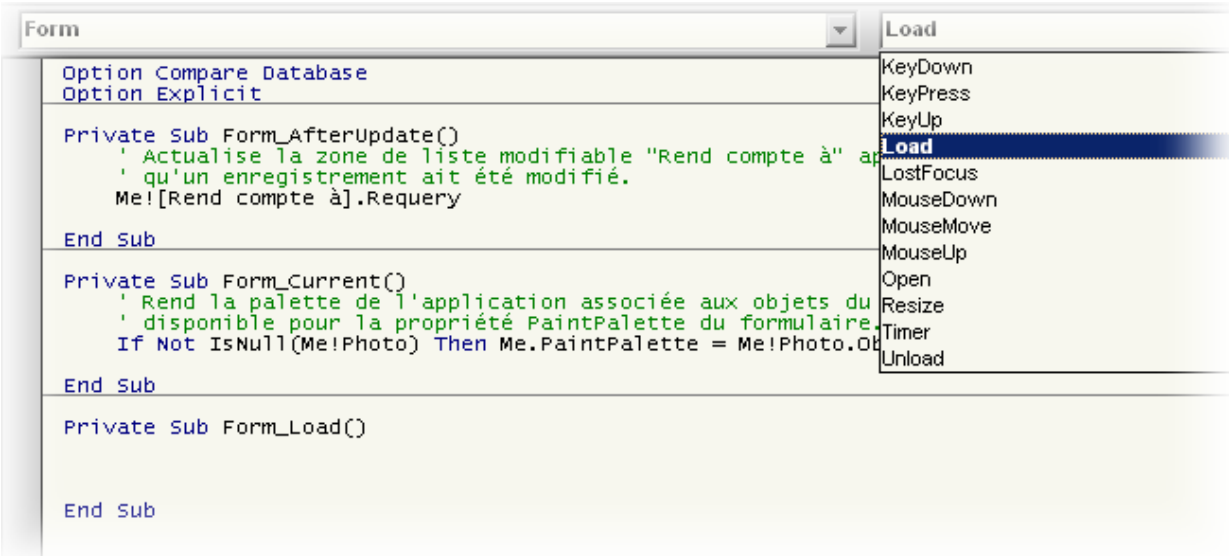
La seule contrainte technique va être de déterminer l'unicité d'un enregistrement.

Dans notre cas, nous testons cette fonctionnalité sur le formulaire **Employés**.



Modification du formulaire Employés

Nous allons placer ce code dans l'événement **Load** (chargement) du formulaire.



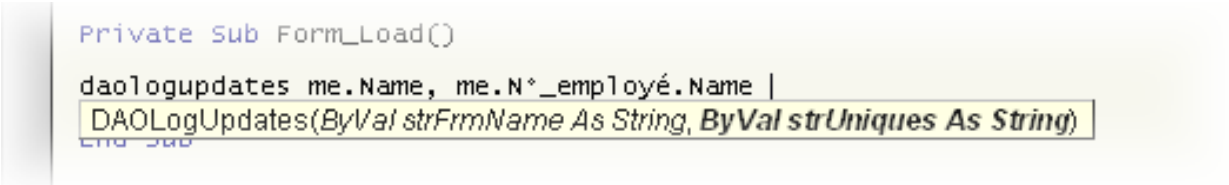
Création de l'événement Load lié au traçage

La fonction comprend deux arguments :

. le **nom du formulaire** : Me.Name

. la **liste des contrôles** qui déterminent un enregistrement unique, si plusieurs séparés par le caractère spécial *pipe* (|).

Me.N°Employé.Name



le code d'appel au traçage en une ligne

VI - Gestion technique des événements de mise à jour

Notre code va s'articuler autour des fonctions suivantes :

Fonction publique appelée pour effectuer le traçage : **DAOLogUpdates**

. attribution dynamique d'événements au formulaire ;

. chargement en mémoire d'un tableau de détermination d'enregistrements uniques.

Par ce code les événements **BeforeUpdate**, **OnDelete** et **AfterInsert** déclencheront nos routines qui sont respectivement : **Form_BeforeUpdate**, **Form_Delete** et **Form_AfterInsert**.

fonction DAOLogUpdates - initialisation du traçage

```
'''""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""'''
'''            module de gestion de log de mise à jour
'''""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""'''
''' @Auteur     :   Charles A. [cafeine]
''' @Projet     :   modLogUpdatesDAO
''' @Version    :   0.2
''' @Date       :   21-12-2005
'''""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""'''
''' @Desc       :   Permet de stocker les mises à jour sur une
'''                 table via n'importe quel formulaire.
'''""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""'''

Private Type Tableau
    Items() As String
End Type

Private Type TableauUniquesItem
    Name As String
    type As Long
    OrdPos As Long
    Control As String
End Type

Private Type TableauUniques
    Items() As TableauUniquesItem
End Type

Private Uniques As TableauUniques

Public Function DAOLogUpdates(ByVal strFrmName As String, _
                              ByVal strUniques As String)
''' active le tracking de modif sur un formulaire
''' argument  :   strFrmName As String (chaîne contenant le nom du formulaire à tracer)
''' argument  :   strUniques as String (chaîne contenant les noms des contrôles qui servent à
identifier un
'''                                    enregistrement unique)
Dim strTemp() As String
Dim iInt As Integer
Dim recBackUp As DAO.Recordset

' vérification de la présence de la table
If Not isTable("tblLogUpdates") Then
    CreateTableLogUpdates
End If

' analyse de la source du formulaire
If Len(Forms(strFrmName).RecordSource) > 0 Then
    ' formulaire basé sur une table  / requête / SQL
    ' attribution des fonctions de traçage sur les événements qui nous intéressent
```

fonction DAOLogUpdates - initialisation du traçage

```
        Forms(strFrmName).BeforeUpdate = "=Form_BeforeUpdate(""" & strFrmName & """)"
        Forms(strFrmName).OnDelete = "=Form_Delete(""" & strFrmName & """)"
        Forms(strFrmName).AfterInsert = "=Form_AfterInsert(""" & strFrmName & """)"

        ' stockage du tableau des Uniques
        Set recBackUp = Forms(strFrmName).RecordsetClone
        strTemp = Split(strUniques, "|")

        For iInt = 0 To UBound(strTemp)
            ReDim Preserve Uniques.Items(iInt)
            Uniques.Items(iInt).Name = Forms(strFrmName).Controls(strTemp(iInt)).ControlSource
            Uniques.Items(iInt).type = recBackUp.Fields(Uniques.Items(iInt).Name).type
            Uniques.Items(iInt).OrdPos = recBackUp.Fields(Uniques.Items(iInt).Name).OrdinalPosition
            Uniques.Items(iInt).Control = strTemp(iInt)
        Next iInt
        recBackUp.Close

        Set recBackUp = Nothing
    Else
        ' formulaire indépendant : traçage impossible
    End If

    End Function
```

VII - Fonction de gestion des événements

VII-A - Evénement Après Insertion (AfterInsert)

Stocker l'événement insertion peut se résumer à récupérer les valeurs des champs clé pour le nouvel enregistrement.

L'annulation consistera a effacer le nouvel enregistrement grâce aux valeurs des clés.

Pour cela nous parcourons le tableau **Uniques**.

Fonction Form_AfterInsert

```
Public Function Form_AfterInsert(ByVal strFrmName As String)
''' fonction exécutée après insertion d'un enregistrement sur le formulaire
'''     assure le stockage des valeurs qui permettent de déterminer l'unicité
'''     de l'enregistrement et ainsi de pouvoir le supprimer
''' argument    :   strFrmName As String (chaîne nom du formulaire)

Dim strUpdate As String
Dim iInt As Integer

strUpdate = "INSERT"
' parcours du tableau des contrôles uniques
For iInt = 0 To UBound(Uniques.Items)
    strUpdate = strUpdate & Uniques.Items(iInt).OrdPos & "=" & _
                SQLTypeDelimiters(Nz(Forms(strFrmName).Controls(Uniques.Items(iInt).Control).Value,
""), Uniques.Items(iInt).type) & ", "
Next iInt
strUpdate = Left(strUpdate, Len(strUpdate) - 2)

' stockage dans la table
AddLogToTable strFrmName, strUpdate, strUpdate

End Function
```

VII-B - Evénement Avant Mise à Jour (BeforeUpdate)

La logique est, cette fois, toute autre.

Nous devons certes stocker les identifiants de l'enregistrement modifié au travers des champs du tableau **Uniques**, mais nous devons en outre enregistrer les anciennes valeurs pour les champs qui ont été modifiés.

C'est la propriété **.OldValue** disponible uniquement sur l'événement BeforeUpdate qui va nous permettre de connaître les champs modifiés.

On travaille sur l'objet RecordsetClone du formulaire, qui comme son nom l'indique est le reflet exact du jeu d'enregistrements source du formulaire.

Fonction Form_BeforeUpdate

```
Public Function Form_BeforeUpdate(ByVal strFrmName As String)
''' fonction exécutée avant la mise à jour d'un enregistrement sur le formulaire
```

Fonction Form_BeforeUpdate

```
    '''     assure le stockage des valeurs qui permettent de déterminer l'unicité
    '''     de l'enregistrement ainsi que les anciennes valeurs et ce afin de pouvoir
    '''     annuler ces mises à jour
    ''' argument    :   strFrmName As String (chaîne nom du formulaire)

    Dim fld As Field
    Dim recBackUp As DAO.Recordset
    Dim strUpdate As String
    Dim strUnique As String
    Dim strControl As String
    Dim iInt As Integer

    ' travail sur le clone du jeu d'enregistrements du formulaire
    Set recBackUp = Forms(strFrmName).RecordsetClone

    If Len(Forms(strFrmName).Controls(Uniques.Items(0).Control).OldValue) > 0 Then
        strUpdate = "UPDATE"
        For Each fld In recBackUp.Fields
            ' nous cherchons le contrôle dont la source est le nom du champ de notre recordset
            '  au moyen de la fonction GetControlBySourceFieldName
            strControl = GetControlBySourceFieldName(fld.Name, strFrmName)
            If Len(strControl) > 0 Then
                ' stockage des anciennes valeurs si changement
                If fld.type <> dbLongBinary Then ' on ne trace pas les champs OLE ... trop long ...
                    ' on ne stocke que les champs modifiés ...
                    If Nz(Forms(strFrmName).Controls(strControl).OldValue, "") <>
    Nz(Forms(strFrmName).Controls(strControl).Value, "") Then
                        strUpdate = strUpdate & fld.OrdinalPosition & "=" & _
    SQLTypeDelimiters(Nz(Forms(strFrmName).Controls(strControl).OldValue, ""), fld.type) & ", "
                    End If
                End If
            End If
        Next fld
        strUpdate = Left(strUpdate, Len(strUpdate) - 2)
        ' ajout des informations permettant de déterminer l'unicité d'un enregistrement
        strUnique = "UNIQUE"
        For iInt = 0 To UBound(Uniques.Items)
            strUnique = strUnique & Uniques.Items(iInt).OrdPos & "=" & _
                        SQLTypeDelimiters(Nz(Forms(strFrmName).Controls(Uniques.Items(iInt).Name).Value),
    _
                                          Uniques.Items(iInt).type) & ", "
        Next iInt
        strUnique = Left(strUnique, Len(strUnique) - 2)

        ' stockage dans la table
        AddLogToTable strFrmName, strUpdate, strUnique

    End If
    recBackUp.Close

    Set recBackUp = Nothing
    Set fld = Nothing

    End Function
```

VII-C - Evénement Sur Effacement (OnDelete)

Il s'agit ici de stocker purement et simplement toutes les valeurs de l'ancien enregistrement.

Nous parcourons ici le **RecorsetClone** du formulaire, et nous nous déplaçons parmi les enregistrements jusqu'au **CurrentRecord**.

Fonction Form_Delete

```
    Public Function Form_Delete(ByVal strFrmName As String)
    ''' fonction exécutée après lors de la suppression d'un enregistrement sur le formulaire
    '''     assure le stockage des valeurs qui permettent de déterminer l'unicité
```

Fonction Form_Delete

```
'''     de l'enregistrement ainsi que les anciennes valeurs et ce afin de pouvoir
'''     annuler ces mises à jour en réinsérant un enregistrement
''' argument    :   strFrmName As String (chaîne nom du formulaire)

Dim fld As Field
Dim recBackUp As DAO.Recordset
Dim strUpdate As String

Set recBackUp = Forms(strFrmName).RecordsetClone

recBackUp.MoveFirst
recBackUp.Move Forms(strFrmName).CurrentRecord - 1

strUpdate = "DELETE"

For Each fld In recBackUp.Fields
    strUpdate = strUpdate & fld.OrdinalPosition & "=" & SQLTypeDelimiters(Nz(fld.Value, ""),
fld.type) & ", "
Next fld
strUpdate = Left(strUpdate, Len(strUpdate) - 2)
recBackUp.Close

Set recBackUp = Nothing
Set fld = Nothing

' stockage dans la table
AddLogToTable strFrmName, strUpdate, ""

End Function
```

VII-D - Fonctions Annexes

La fonction **SQLTypeDelimiters** fonctionne un peu comme la fonction native **BuildCriteria()** et nous permet de renvoyer une chaîne avec un typage des données au format de l'implémentation SQL d'Access.

Fonction SQLTypeDelimiters

```
Private Function SQLTypeDelimiters(ByVal varValue, _
                                   ByVal intType As Long) As String
''' fonction renvoyant une chaîne compatible avec l'implémentation SQL d'access
'''     en fonction du type du champ
''' argument    :   varValue (valeur du champ)
''' argument    :   intType As Long (code du type de champ)

Select Case intType
    Case -1
        SQLTypeDelimiters = vbNullString

    Case dbBigInt, dbBinary, dbBoolean, dbByte, dbCurrency, _
         dbDecimal, dbDouble, dbFloat, dbGUID, dbInteger, _
         dbLong, dbNumeric, dbSingle
        ' types de numériques : pas de formatage
        If Len(varValue) > 0 Then
            SQLTypeDelimiters = varValue
        Else
            SQLTypeDelimiters = "Null"
        End If

    Case dbChar, dbMemo, dbText
        ' types de chaînes : utilisation des doubles quotes (")
        SQLTypeDelimiters = """" & Replace(varValue, Chr(34), Chr(34) & Chr(34)) & """"

    Case dbDate, dbTime, dbTimeStamp
        ' types de dates : utilisation des dièses (#) et du format de date US
        If Len(varValue) > 0 Then
            SQLTypeDelimiters = "#" & Format(CDate(varValue), "mm/dd/yyyy hh:nn:ss") & "#"
        Else
            SQLTypeDelimiters = vbNullString
        End If
End Select
```

Fonction SQLTypeDelimiters

```
    End Function
```

La fonction **GetControlBySourceFieldName** permet de retrouver le nom d'un contrôle d'un formulaire par sa source.

Fonction GetControlBySourceFieldName

```
    Private Function GetControlBySourceFieldName(ByVal strSource As String, _
                                                 ByVal strFrmName As String) As String
    ''' fonction renvoyant une chaîne qui est le nom du contrôle correspondant à un champ d'une
    '''     source de données d'un formulaire
    ''' argument    :   strSource (valeur du champ)
    ''' argument    :   intType As Long (code du type de champ)

    Dim ctl As Control

    On Error GoTo NextCtlSearch

    For Each ctl In Forms(strFrmName).Controls
        If ctl.ControlSource = strSource Then
            GetControlBySourceFieldName = ctl.Name
            Exit Function
        End If
    NextItem:
    Next ctl

    GetControlBySourceFieldName = vbNullString
    Set ctl = Nothing
    Exit Function

    NextCtlSearch:
        Err.Clear
        Resume NextItem
    End Function
```

La fonction **AddToLogTable** insère le log des modifications dans notre table **tblLogUpdates**.

Procédure AddLogToTable

```
    Private Sub AddLogToTable(ByVal strFrmName As String, _
                              ByVal strAction As String, _
                              ByVal strUnique As String)
    ''' procédure prenant en charge l'ajout d'informations à la table de traçage
    ''' argument    :   strFrmName As String (chaîne nom du formulaire)
    ''' argument    :   strAction As String (chaîne des champs mis à jour)
    ''' argument    :   strUnique As String (chaîne des champs déterminant un enregistrement unique)

    Dim strSQL As String

    strSQL = "INSERT INTO tblLogUpdates " & _
                "(author, Type, Lot, form, revertAction, revertUnique, datUpdate, selection) " & _
                "VALUES (""" & GetLogin() & """, """ & _
                                Left(strAction, 6) & """, """ & _
                                "L" & Format(Now, "yymmddhhnnss") & """, """ & _
                                strFrmName & """, """ & _
                                Replace(strAction, Chr(34), Chr(34) & Chr(34)) & """, """ & _
                                Replace(strUnique, Chr(34), Chr(34) & Chr(34)) & """, #" & _
                                Format(Now, "mm/dd/yyyy hh:nn:ss") & "#, " & _
```

Procédure AddLogToTable

```
                                """0"");"
    DoCmd.RunSQL strSQL

    End Sub
```

VIII - Consulter ou Annuler des mises à jour

Les mises à jour sont maintenant enregistrées dans la table tblLogUpdates.

Il nous faut créer un formulaire de consultation / recherche des mises à jour et mettre en place les fonctionnalités d'annulation.

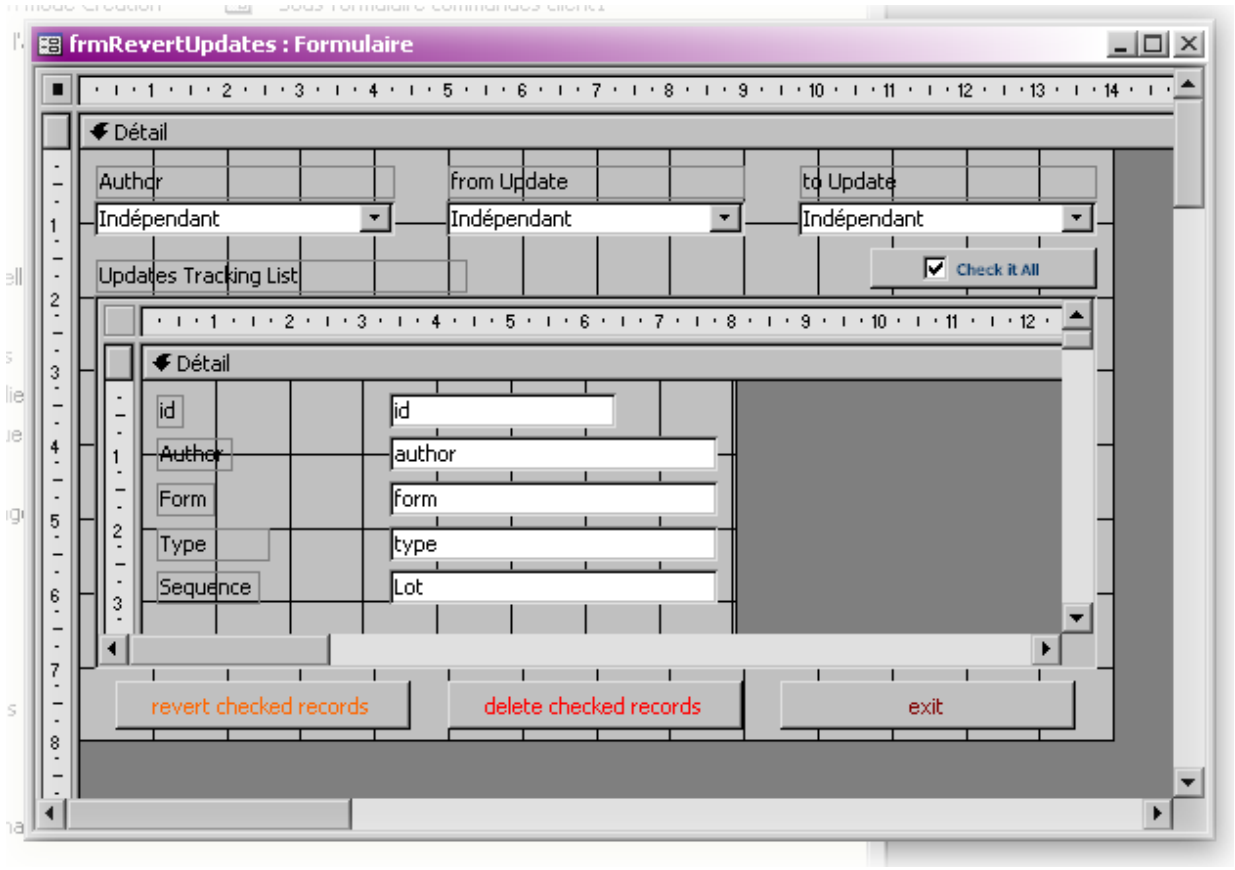
VIII-A - Formulaire de consultation

Nous avançons en territoire connu, il s'agit d'un classique formulaire de recherche multi-critères, pour plus d'information n'hésitez pas plus longtemps à lire mon tutoriel sur le sujet.

Il est composé de :

. un sous-formulaire en mode feuille de données qui va comprendre les mises à jour ;

. un formulaire principal qui contiendra les critères de sélection.



Formulaire de recherche et annulation de mises à jour

Fonctionnalités de consultation

```
Private Sub Form_Load()
' rechargement des critères sur chaque modification d'un contrôle de recherche
'  grâce à la Fonction RefreshSubForm

Dim ctl As Control

For Each ctl In Me.Controls
    If ctl.ControlType = acComboBox Then
        ctl.AfterUpdate = "=RefreshSubForm()"
    End If
Next ctl

Set ctl = Nothing

RefreshSubForm

End Sub

Private Function RefreshSubForm()
' fonction de mise à jour de recherche multi-critères sur la
'  table des updates

Dim strSQL As String

strSQL = "SELECT * FROM tblLogUpdates " & _
         "WHERE author Like '*" & Me.cmbAuthor & "*' And "

If Me.cmbLot0 = "*" Then
```

Fonctionnalités de consultation
```
        If Me.cmbLot1 = "*" Then
            strSQL = Left(strSQL, Len(strSQL) - 5)
        Else
            strSQL = strSQL & "datUpdate <= #" & Format(Me.cmbLot1, "mm/dd/yyyy hh:nn:ss") & "#"
        End If
    Else
        If Me.cmbLot1 = "*" Then
            strSQL = strSQL & "datUpdate >= #" & Format(Me.cmbLot0, "mm/dd/yyyy hh:nn:ss") & "#"
        Else
            strSQL = strSQL & "datUpdate BETWEEN #" & Format(Me.cmbLot0, "mm/dd/yyyy hh:nn:ss") & _
                              "# AND #" & Format(Me.cmbLot1, "mm/dd/yyyy hh:nn:ss") & "#"
        End If
    End If
    Me.subfrmRevertUpdates.Form.RecordSource = strSQL & " ORDER BY datUpdate DESC;"

    Me.subfrmRevertUpdates.Form.Requery

    End Function
```

Un des points intéressants du code est l'utilisation du code source de la FAQ, pour ajouter le choix "Tous" à un ComboBox.

SQL d'ajout du choix 'tous' ici pour le ComboBox Author
```
      SELECT tblLogUpdates.author
      FROM tblLogUpdates
      GROUP BY tblLogUpdates.author
      ORDER BY tblLogUpdates.author
    UNION
      SELECT TOP 1 "*"
      FROM tblLogUpdates;
```

VIII-B - Fonctionnalités d'annulation

La consultation du formulaire **frmRevertUpdates** permet de pointer sur une modification précise.

Une des colonnes cachées du sous-formulaire **subfrmRevertUpdates** contient l'ID.

Il nous suffira de créer une fonction d'annulation avec comme argument cet ID.

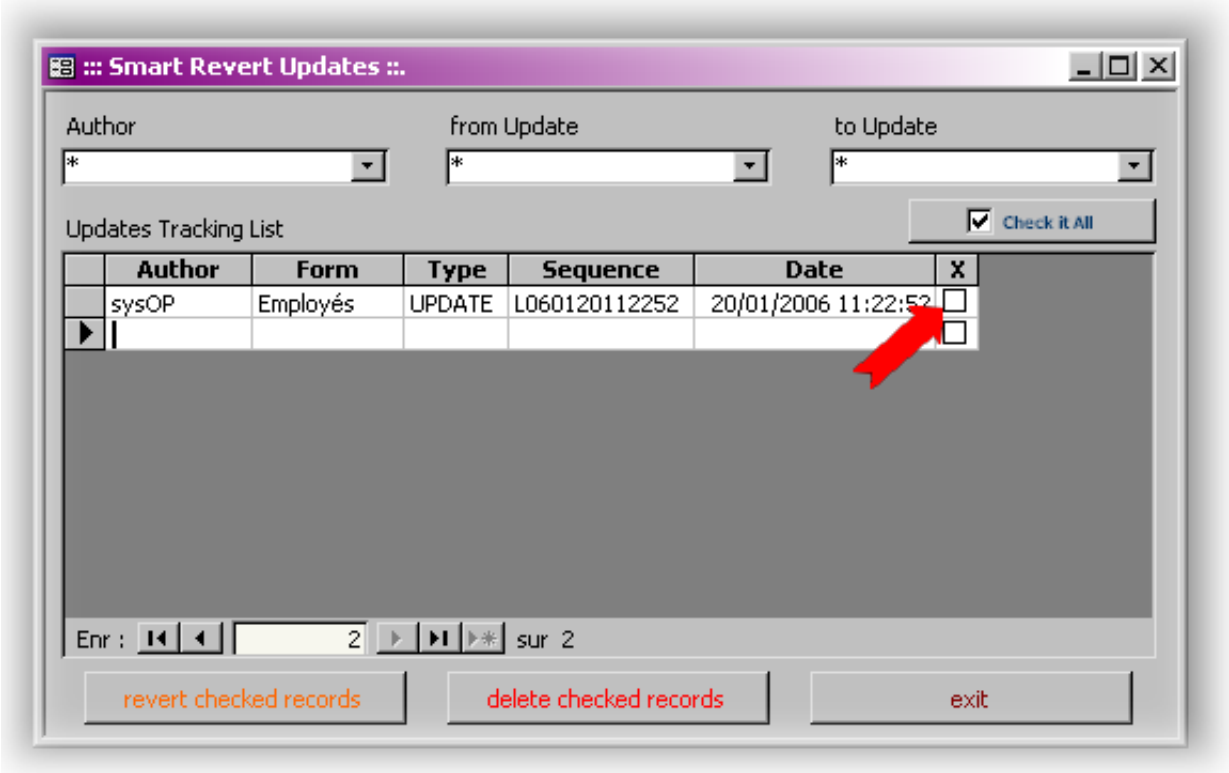
Bouton d'annulation du formulaire
```
    Private Sub cmdRevert_Click()

    Dim rec As DAO.Recordset
    Dim db As DAO.Database

    RefreshSubForm

    Set db = CurrentDb()

    Set rec = db.OpenRecordset(Me.subfrmRevertUpdates.Form.RecordSource, dbOpenSnapshot)

    ' Parcours du recordset de la table des Updates
    '  appel de la fonction DAORevertUpdate dans le cas où
    '  l'item a été sélectionné.
    Do While Not rec.EOF
        If rec!selection Then
            DAORevertUpdate rec!id
        End If
```

Bouton d'annulation du formulaire

```
        rec.MoveNext
    Loop

    rec.Close

    Set rec = Nothing
    Set db = Nothing

    Me.subfrmRevertUpdates.Form.Requery

    End Sub
```

Procédure d'annulation d'une mise à jour

```
    Public Sub DAORevertUpdate(ByVal idOperation As Long)
    ''' fonction prenant en charge l'annulation d'une mise à jour
    ''' argument    :   idOperation As Long (identifiant de la mise à jour dans la table tblLogUpdates)
    '''""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
    ''' fonctions   :   Exécution d'une mise à jour d'annulation
    '''                 Suppression de l'enregistrement dans la table tblLogUpdates

    Dim db As DAO.Database
    Dim rec As DAO.Recordset

    Set db = CurrentDb()
    Set rec = db.OpenRecordset("SELECT * FROM tblLogUpdates WHERE id = " & idOperation, dbOpenSnapshot)

    If Len(rec!revertAction) > 0 Then
            ' Appel de la fonction d'annulation proprement dite.
        RevertActionString rec!Form, rec!revertAction, rec!revertUnique
        ' Effacement de l'item après annulation.
        DoCmd.RunSQL "DELETE * FROM tblLogUpdates WHERE id = " & idOperation & ";"
    End If

    rec.Close
    Set rec = Nothing
    Set db = Nothing

    End Sub
```



Sous-formulaire d'annulation

Procédure d'annulation DAO

```
Private Sub RevertActionString(ByVal strFrmName As String, _
                               ByVal strAction As String, _
                               ByVal strUniques As String)

Dim tabRevert As Tableau
Dim tabUniques As Tableau
Dim strDummy As String
Dim strCrit As String
Dim iInt As Integer
Dim blnWasOpen As Boolean
Dim recRevert As DAO.Recordset

On Error GoTo RASerrHandler

'tentative d'accès au formulaire
strDummy = Forms(strFrmName).Name
' si pas d'erreur, cela signifie que le formulaire était déjà ouvert
' nous passons le flag d'ouverture de form à False
blnWasOpen = True

RASformAvailable:
Set recRevert = Forms(strFrmName).RecordsetClone

' chargement du tableau de revert
tabRevert = LoadRevertActionString(strAction, recRevert.Fields.Count)
tabUniques = LoadRevertActionString(strUniques, UBound(Uniques.Items))

Select Case Left(strAction, 6)
    Case "UPDATE"
        ' cas d'une mise à jour d'enregistrement
        ' l'annulation va simplement rétablir les anciennes valeurs
        For iInt = 0 To UBound(Uniques.Items)
            strCrit = strCrit & "[" & Uniques.Items(iInt).Name & "] = " & _
                      SQLTypeDelimiters(tabUniques.Items(iInt), Uniques.Items(iInt).type) & " AND "
```

Procédure d'annulation DAO

```
        Next iInt
        strCrit = Left(strCrit, Len(strCrit) - 5)

        recRevert.FindFirst strCrit
        If Not recRevert.NoMatch Then
            recRevert.Edit
            For iInt = 0 To recRevert.Fields.Count - 1
                If tabRevert.Items(iInt) <> """""" And Len(tabRevert.Items(iInt)) > 0 Then
                    If Left(tabRevert.Items(iInt), 1) = "#" Then
                        recRevert.Fields(iInt).Value = CDate(Replace(tabRevert.Items(iInt), "#",
vbNullString))
                    ElseIf Left(tabRevert.Items(iInt), 1) = """" Then
                        ' cas de texte, nous remplaçons les doubles quotes par rien
                        recRevert.Fields(iInt).Value = Replace(Mid(tabRevert.Items(iInt), 2,
Len(tabRevert.Items(iInt)) - 2), _
                                                               Chr(34) & Chr(34), _
                                                               Chr(34))

                    ElseIf tabRevert.Items(iInt) = "Null" Then
                        recRevert.Fields(iInt).Value = Null
                    Else
                        recRevert.Fields(iInt).Value = tabRevert.Items(iInt)
                    End If
                End If
            Next iInt
            recRevert.Update
        Else
            MsgBox "Update non revertable, record not found", vbCritical + vbOKOnly, "Update Revert"
        End If

    Case "INSERT"
        ' cas d'un ajout d'enregistrement
        ' l'annulation va simplement consister à effacer ce nouvel enregistrement
        For iInt = 0 To UBound(Uniques.Items)
            strCrit = strCrit & "[" & Uniques.Items(iInt).Name & "] = " & _
                      SQLTypeDelimiters(tabUniques.Items(iInt), Uniques.Items(iInt).type) & " AND "
        Next iInt
        strCrit = Left(strCrit, Len(strCrit) - 5)
        recRevert.FindFirst strCrit
        If Not recRevert.NoMatch Then
            recRevert.Delete
        Else
            MsgBox "Update non revertable, record not found", vbCritical + vbOKOnly, "Update Revert"
        End If

    Case "DELETE"
        ' cas d'une suppression d'enregistrement
        ' l'annulation va simplement consister à ajouter à nouveau cet enregistrement
        ' code Ok
        recRevert.AddNew
        For iInt = 0 To recRevert.Fields.Count - 1
            If tabRevert.Items(iInt) <> """""" And Len(tabRevert.Items(iInt)) > 0 Then
                If Left(tabRevert.Items(iInt), 1) = "#" Then
                    recRevert.Fields(iInt).Value = CDate(Replace(tabRevert.Items(iInt), "#",
vbNullString))
                ElseIf Left(tabRevert.Items(iInt), 1) = """" Then
                    ' cas de texte, nous remplaçons les doubles quotes par rien
                    recRevert.Fields(iInt).Value = Replace(Mid(tabRevert.Items(iInt), 2,
Len(tabRevert.Items(iInt)) - 2), _
                                                           Chr(34) & Chr(34), _
                                                           Chr(34))

                ElseIf tabRevert.Items(iInt) = "Null" Then
                    recRevert.Fields(iInt).Value = Null
                Else
                    recRevert.Fields(iInt).Value = tabRevert.Items(iInt)
                End If
            End If
        Next iInt
        recRevert.Update

End Select

recRevert.Close

Set recRevert = Nothing

If Not blnWasOpen Then
```

Procédure d'annulation DAO

```
        DoCmd.Close acForm, strFrmName
    Else
        Forms(strFrmName).Requery
    End If

    Exit Sub

    RASerrHandler:
    Select Case Err.Number
        Case "3164"
            ' le champ ne peut être mis à jour
            Err.Clear
            Resume Next
        Case "2450"
            ' le formulaire n'est pas ouvert
            ' nous l'ouvrons en mode caché
            DoCmd.OpenForm strFrmName, acNormal, , , , acHidden
            Err.Clear
            ' nous passons le flag d'ouverture de form à False
            blnWasOpen = False
            Resume RASformAvailable
        Case Else
            MsgBox Err.Number & vbCrLf & Err.Description
            Resume Next

    End Select

    End Sub
```

La procédure qui charge le tableau d'annulation à partir d'une chaîne.

Chargement de la chaîne d'annulation

```
    Private Function LoadRevertActionString(ByVal strAction As String, _
                                            ByVal intFieldsCount As Integer) As Tableau

    Dim strTemp() As String
    Dim strTemp2() As String
    Dim strItem As String
    Dim iInt As Integer

    If Len(strAction) > 0 Then
        strTemp = Split(Right(strAction, Len(strAction) - 6), ", ")
        ReDim LoadRevertActionString.Items(intFieldsCount)
        For iInt = 0 To UBound(strTemp)
            If Len(strTemp(iInt)) > 0 Then
                strTemp2 = Split(strTemp(iInt), "=")
                LoadRevertActionString.Items(strTemp2(0)) = strTemp2(1)
            End If
        Next iInt
    End If

    End Function
```

IX - Conclusion

Cet article n'a d'ambition que d'ouvrir une perspective sur cette problématique.

Nous sommes parvenus à réaliser dans ce court tutoriel une solution de stockage de mises à jour.

De nombreuses autres solutions sont possibles, à vous de les adapter aux besoins de vos applications.

Pour information, j'ai longtemps étudié une solution basée sur le **XML**, mais j'y ai pour l'instant renoncé en raison des difficultés d'import.