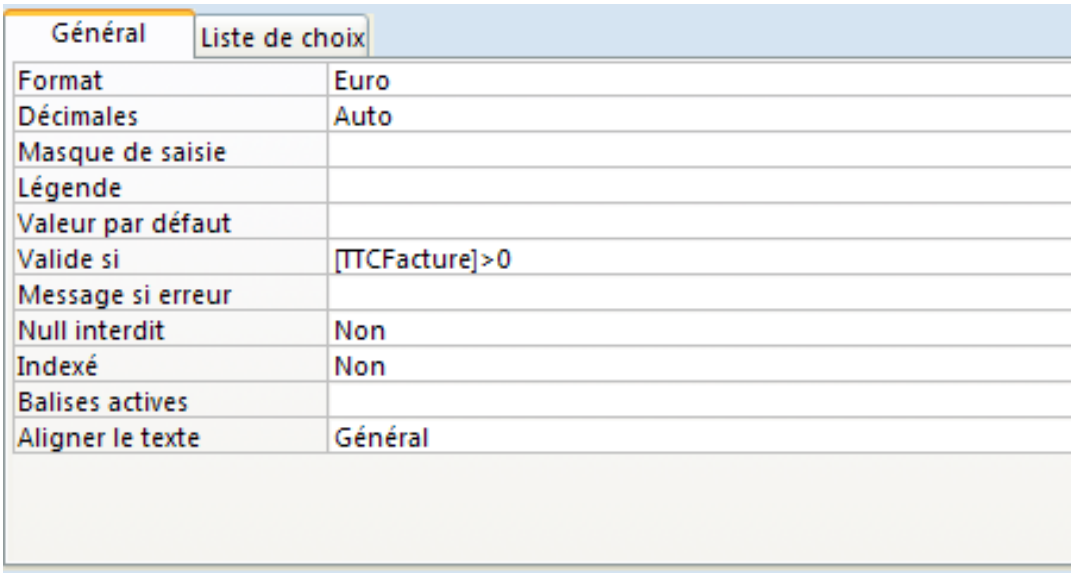
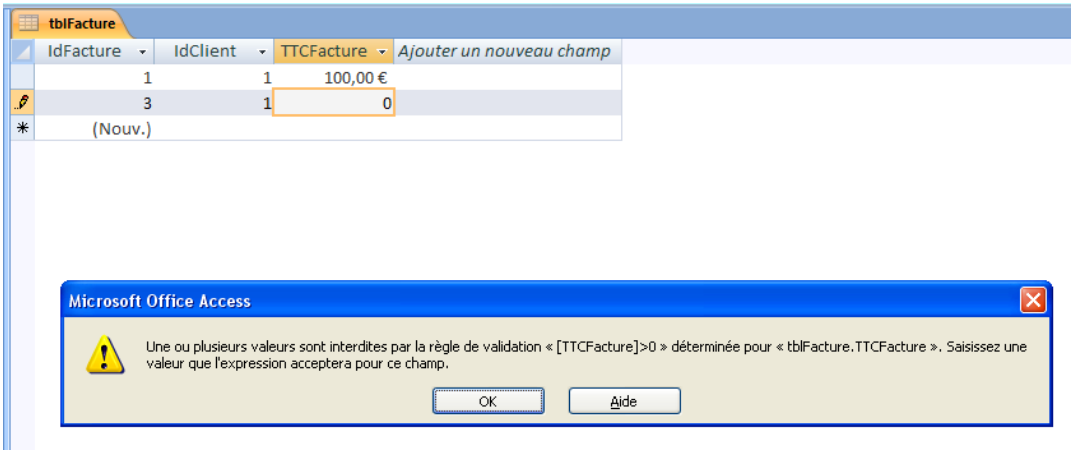
# I - Les contraintes ValideSi et leur limite

Vous connaissez sûrement la propriété **ValideSi** des champs des tables Access. Cette propriété permet de définir une contrainte à valider lors de la création ou la modification de l'enregistrement. Si la contrainte n'est pas vérifiée, l'enregistrement des données est impossible.

Voici un exemple de propriété **ValideSi** concernant le montant total d'une facture dont l'énoncé interdit qu'il soit négatif :




Si l'utilisateur tente une saisie ne correspondant pas à la règle, voici ce qu'il se passe :




*Dans l'exemple, l'utilisateur tente de saisir la donnée directement dans la table. N'ayez pas peur, ce mécanisme de contrainte est aussi valable lorsque l'utilisateur opère sa saisie depuis un formulaire ou que celle-ci est automatisée par une requête ou par du code VBA.*

La propriété **Message si Erreur** permet quant à elle de définir le message qui sera affiché si la contrainte n'est pas respectée. Le but étant d'obtenir quelque chose de plus explicite que l'image ci-dessus. Toutefois, étant donné que dans bien des cas, le développeur gère lui-même ces propres messages d'alerte dans son code VBA, cette propriété dispose donc d'un intérêt limité.

Malheureusement, l'efficacité de la propriété **ValideSi** se limite exclusivement à tester la valeur de la donnée saisie par rapport à une règle de gestion basique. Ainsi, il est possible de vérifier la taille d'un texte, l'appartenance d'une

date à un intervalle, la supériorité d'un nombre à une constante, etc. Par contre il est impossible de faire référence aux données déjà inscrites dans la base.

Prenons l'exemple d'une table **tblLimite** qui définirait le montant maximum autorisé pour la saisie des factures. Cette règle de gestion n'a pas vraiment de sens dans la vie réelle mais permet d'énoncer clairement mes propos.

La définition suivante de la propriété **ValideSi** paraît à priori juste, tout du moins du point de vue du sens :

```
[TTCFacture]<=(SELECT Limite FROM tblLimite)
```

Pourtant, elle est refusée par Microsoft Access.
Pourquoi ? Tout simplement, parce qu'il est impossible de faire référence à une autre donnée issue de la base dans la propriété **ValideSi** d'un champ.

## II - ADODB et CONSTRAINT

Pour passer outre la limitation de la clause **ValideSi** vue plus haut, il est nécessaire de construire la contrainte manuellement à l'aide d'une instruction SQL après la création de la table à l'aide de l'instruction suivante :

*Il est plus intéressant de la faire après la création de la table tout simplement car cela vous permet de bénéficier auparavant des assistants pour mettre en place vos champs,vos clés et vos index.*

```
ALTER TABLE nom de la table
  ADD CONSTRAINT nom de la contrainte
    CHECK (clause de la contrainte)
```

Pour reprendre l'exemple d'une facture avec un montant exclusivement positif :

```
ALTER TABLE tblFacture
  ADD CONSTRAINT MontantPositif
    CHECK (TTCFacture>0)
```

Ou pour le cas de la limitation du montant d'une facture :

```
ALTER TABLE tblFacture
  ADD CONSTRAINT LimiteMontant
    CHECK   (TTCFacture<(
              SELECT limite
              FROM tblLimite
                )
        )
```

Mais ce n'est pas aussi simple. Essayez de lancer cette instruction SQL depuis le générateur de requêtes et vous vous apercevrez qu'elle est refusée sous prétexte qu'une erreur de syntaxe serait présente au sein de la clause **CHECK**. Et pourtant, je peux vous assurer que l'instruction ci-dessus est valide.

Le problème est que le générateur de requête, tout comme l'assistant de création de tables, refuse de faire référence à des données externes à la saisie. Il en est de même pour les codes VBA basés sur DAO.

La solution consiste à lancer la requête de définition de la contrainte en utilisant un accès *Jet OLE DB* depuis une connexion **ADODB**. Pour faire simple, **ADODB** est un mode d'accès aux données assez proche de **DAO** (d'un point de vue structure) mais destiné avant tout à des moteurs de bases de données plus évolués (SQL Server idéalement). Ce n'est pas le cas ici, mais toujours est-il qu'il va permettre de faire ce que l'on souhaite. Il est inutile à ce stade de

savoir comment ouvrir une connexion **ADODB** puisque depuis Access 2000, cette connexion est accessible aisément en VBA à l'aide de la propriété :

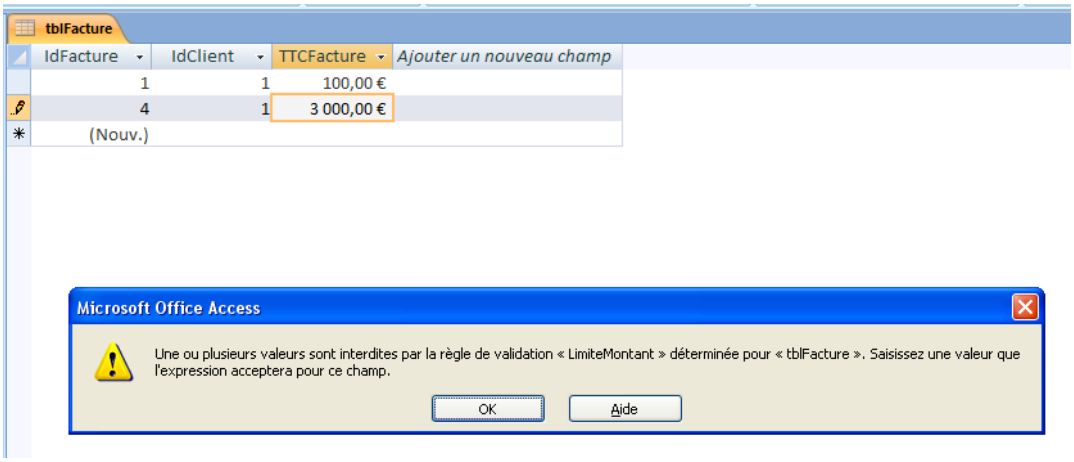
```
Application.CurrentProject.Connection
```

L'objet **Connection** retourné expose une méthode **Execute** permettant de lancer un ordre SQL. C'est donc cette méthode qui sera utilisée pour lancer la requête.

Il suffit donc de lancer le bloc d'instructions suivant depuis un module VBA :

```
Sub Contrainte()
Dim strSQL As String
strSQL = "ALTER TABLE tblFacture ADD CONSTRAINT LimiteMontant " & _
         "CHECK (TTCFacture<(SELECT limite FROM tblLimite))"
Application.CurrentProject.Connection.Execute strSQL
End Sub
```

Si l'utilisateur tente maintenant de saisir un montant supérieur à celui de la limite définie dans la table **tblLimite**, sa saisie est refusée :



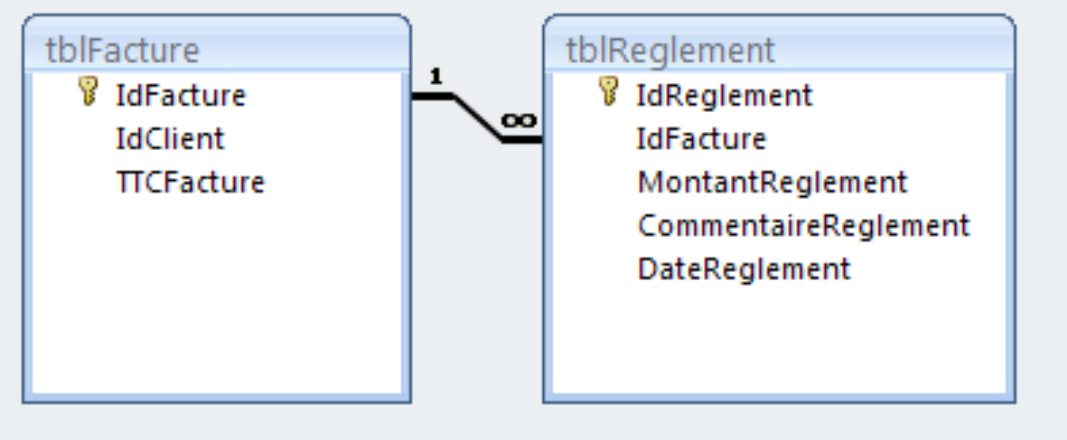

Il existe toutefois quelques limitations :

- il est impossible de changer le message d'erreur affiché, contrairement à la propriété **ValideSi**,
- il est déconseillé d'utiliser en même temps des contraintes CHECK et des propriétés **ValideSi** dans la même table.

## III - Contraintes par jointure

Dans l'exemple plus haut, rien ne lie directement le montant limite au montant TTC de la facture. J'entends par là, que toutes les factures auront le même montant limite autorisé. Pourtant dans bien des cas, il est nécessaire que la contrainte évolue en fonction de la donnée saisie.

Dans le cas d'une table **tblReglement**, en lien avec la table **tblFacture**, il est fort probable qu'il soit interdit de saisir un réglement dont le montant dépasse celui de la facture.




Le code SQL de la contrainte souhaitée est alors :

```
ALTER TABLE tblReglement
  ADD CONSTRAINT ReglementMaxi
    CHECK    (MontantReglement <= (
                  SELECT TTCFacture
                  FROM tblFacture
                  WHERE idFacture=tblreglement.idfacture
                  )
        )
```

La contrainte **CHECK** vérifie alors que le montant du réglement est inférieur ou égal à celui de la facture. Le lien entre la facture concernée par le règlement et l'enregistrement dans la table tblFacture est assuré par la clause WHERE **idFacture=tblreglement.idFacture** qui signifie :

*"La ligne de la table **tblFacture** où l'**idFacture** est égal à l'**idFacture** de la table **tblReglement** en cours de modification "*

## IV - Contraintes calculées

Il s'agit là de la partie la plus complexe de mise en place de contraintes. Prenons comme exemple une contrainte devant répondre à la règle de gestion suivante :

*La somme des règlements d'une facture ne doit pas être supérieure au montant total de la facture.*

Son code SQL serait :

```
ALTER TABLE tblReglement
  ADD CONSTRAINT SommeReglementMaxi
        CHECK    (
          (
          SELECT Sum(MontantReglement)
          FROM tblReglement tblReglementCalcul
          WHERE tblReglementCalcul.idFacture=tblreglement.idfacture
          )
          <=
          (
          SELECT TTCFacture
          FROM tblFacture
          WHERE idFacture=tblreglement.idfacture
          )
        )
```

La clause CHECK est alors composée de deux requêtes :

```
CHECK (requete1<=requete2)
```

La partie de droite est simple, il s'agit de la même que celle vue plus haut. Elle retourne le montant TTC de la facture correspondante au règlement.
Si celle de gauche a aussi un sens très simple : retourner la somme des règlements pour cette facture, son écriture est plus complexe. En effet, lors de la création ou de la modification d'un règlement, le moteur travaille sur la table **tblReglement** et l'enregistrement en cours est composé de *IdReglement, idFacture, MontantReglement, DateReglement, CommentaireReglement*. Rien n'indique les montants déjà réglés pour cette facture à cet instant. Pour connaitre ce montant, il faut accéder à une nouvelle vue de la table **tblReglement** (comme une photocopie). C'est sur cette vue que sera effectué le calcul. Schématiquement, c'est comme si deux tableaux étaient ouverts en même temps. Le stylo de la main droite est prêt à remplir le premier, pendant que l'oeil, aidé de l'index de la main gauche, vérifie que le montant ne dépassera pas la somme restant à payer sur le second tableau. Et comme il est impossible d'ouvrir deux fois le même objet avec le même nom, SQL impose d'utiliser un alias pour le deuxième appel, d'où l'instruction :

```
FROM tblReglement tblReglementCalcul
```

Qui peut aussi s'écrire :

```
FROM tblReglement AS tblReglementCalcul
```

La clause :

```
tblReglementCalcul.idFacture=tblreglement.idfacture
```

Permet ainsi de faire le lien entre les deux vues en se basant sur l'**idFacture**.

## V - Contraintes CHECK (X=TRUE)

La forme **CHECK (X=TRUE)** permet de mettre en place n'importe quelle condition qui ne serait pas basée sur la valeur des données saisies. En fait, il s'agit simplement de tester si la condition à gauche (X) du signe = est vraie. Dès lors toutes les fantaisies sont possibles.

Ainsi, il est possible par exemple de limiter le nombre de lignes dans une table. Dans l'exemple suivant, il est interdit de saisir plus d'un enregistrement dans la table **tblLimite**.

```
ALTER TABLE tblLimite
  ADD CONSTRAINT NbLigneMaxi
    CHECK   (
            (
            SELECT Count(limite)<2
            FROM tblLimite
            )
            =TRUE
          )
```

Et puis, pourquoi pas, ne pas interdire la modification des données tous les lundis :

```
ALTER TABLE tblLimite
  ADD CONSTRAINT InterditLundi
    CHECK (Weekday(date())<>2=TRUE )
```

Comme vous le voyez, la puissance du code SQL permet de satisfaire quasiment à toutes les exigences. Attention cependant : il s'agit de code SQL Access et non de code VBA. Vous ne pouvez donc pas faire appel aux éléments propres à VBA, tels que les fonctions personnalisées d'un module par exemple. De plus sous Access 2007, les champs à valeurs multiples ne peuvent pas intégrer la clause CHECK.

## VI - Suppression des contraintes

Les contraintes ainsi créées n'étant pas répertoriées dans l'interface graphique d'Access, il est nécessaire, pour les supprimer, d'utiliser la même méthode que pour leur création, à savoir une requête SQL exécutée avec une connexion ADO DB. La syntaxe de l'ordre SQL est la suivante :

```
ALTER TABLE <nomdelatable> DROP <nomdelacontrainte>
```

Ce qui donne en VBA :

```
Sub SuppressionContrainte()
Dim oCnx As ADODB.Connection
Dim strSQL As String
Set oCnx = Application.CurrentProject.Connection
strSQL = "ALTER TABLE tblLimite DROP CONSTRAINT LimiteMontant"
oCnx.Execute strSQL
End Sub
```

*Pour pouvoir supprimer une table, il est nécessaire de supprimer toutes les contraintes s'y rattachant.*

## VII - Modification d'une contrainte

La syntaxe :

```
ALTER TABLE <nomdelatable> ALTER  CONSTRAINT ...
```

n'étant pas reconnue, il est impossible de modifier le code de définition d'une contrainte. La solution : **supprimer** et **recréer**.

## VIII - Capture dans un formulaire

### VIII-A - Affichage personnalisé

Comme expliqué au début de ce document, lorsque les contraintes sont définies dans une clause **CHECK** en SQL et non via la propriété **ValideSi**, il est impossible de définir le message d'erreur à afficher. Cependant lorsque la saisie est opérée depuis un formulaire, l'évènement Sur Erreur (**OnError**) est levé dès qu'une contrainte n'est pas respectée et le code d'erreur correspondant est le n°**3317**. Vous pouvez donc utiliser cet évènement pour personnaliser l'affichage.

```
Private Sub Form_Error(DataErr As Integer, Response As Integer)
Select Case DataErr
    Case 3317 ' erreur de validation de containte
        MsgBox "Une contrainte n'est pas validée"
        ' Continue sans afficher l'erreur standard
        Response = acDataErrContinue
    Case Else
        Response = acDataErrDisplay
End Select
```

```
End Sub
```

Malheureusement, comme vous pouvez le constater l'exemple de code ci-dessus affiche qu'une contrainte n'est pas respectée mais est incapable d'énoncer laquelle. A ce stade de l'exécution, bien que l'évènement **OnError** du formulaire soit déclenché aucune information concernant l'erreur n'a été envoyée au code qui s'exécute. C'est seulement quand l'événement **OnError** sera terminé que la gestion d'erreur en VBA va pouvoir être utilisée via un objet **Err**. L'astuce, provenant de *Thierry Gasperment*, consiste donc à démarrer la minuterie du formulaire dans l'évènement **OnError** de telle sorte à capturer l'erreur à sa sortie en tentant une nouvelle fois d'enregistrer les données.

```
Private Sub Form_Error(DataErr As Integer, Response As Integer)
Select Case DataErr
    Case 3317 ' erreur de validation de containte
        ' Lance la minuterie
        Me.TimerInterval = 1
        ' Continue sans afficher l'erreur standard
        Response = acDataErrContinue
    Case Else
        Response = acDataErrDisplay
End Select
End Sub



Private Sub Form_Timer()
    ' Arrête la minuterie
    Me.TimerInterval = 0

On Error GoTo gestion_erreurs
    ' Tente d'enregistrer les données
    Me.Dirty = False
    Exit Sub


gestion_erreurs:
' Si une erreur est lancée, lance la procédure GestionErreur()
GestionErreur Err.Description
End Sub
```

Voici le code de la procédure de gestion d'erreur :

```
Public Sub GestionErreur(pMessage As String)
Dim lPosArg1 As Long
Dim lPosArg2 As Long
Dim lLenRule As Long
Dim lRule As String
Dim lNewMessage As String
On Error GoTo gestion_erreurs
' Position des arguments dans le message standard
lPosArg2 = InStr(1, AccessError(3317), "|2")
lPosArg1 = InStr(1, AccessError(3317), "|1")
If lPosArg2 > 0 Then
    ' Taille du nom de la règle
    lLenRule = InStr(1, pMessage, Mid(AccessError(3317), lPosArg2 + 2, lPosArg1 - lPosArg2 - 2)) -
 lPosArg2
    ' Lit le nom de la règle en erreur
    lRule = Trim(Mid(pMessage, lPosArg2, lLenRule))
    ' Construit le message personnalisé
    lNewMessage = "La règle " & lRule & " n'est pas validée :"
    ' Ajout de texte en fonction de la règle non validée
    Select Case lRule
        Case "ReglementMaxi"
            lNewMessage = lNewMessage & vbCrLf & "Le règlement dépasse le montant de la facture"
        Case "SommeReglementMaxi"
            lNewMessage =
 lNewMessage & vbCrLf & "La somme des réglements dépasse le montant de la facture"
```

```
    End Select
    ' Affiche le message personnalisé
    MsgBox lNewMessage
Else
    ' Au cas où, affiche le message sans traitement
    MsgBox pMessage
End If
Exit Sub
gestion_erreurs:
MsgBox "Erreur n° " & Err.Number & ", " & Err.Description
End Sub
```

Le principe de cette procédure est simple. Il consiste à découper le message d'erreur pour y récupérer le nom de la contrainte. Pour cela, il faut s'appuyer sur le message d'erreur d'origine défini par la fonction **AccessError**. Dans ce message d'origine, le nom de la contrainte est remplacé par le paramètre " **|2** ". Il suffit donc de localiser ce paramètre dans le message d'erreur complet passé à la fonction pour être en mesure de définir la variable **lRule** qui contiendra, ainsi, le nom de la contrainte ayant levé l'erreur.

Enfin, la structure **SELECT CASE** permet de personnaliser le message à afficher en fonction du nom de la contrainte trouvé.

## VIII-B - Afficher la définition de la contrainte

La propriété **CheckConstraints** du recordset d'un formulaire lié à une source de données regroupe la liste des contraintes séparées par le caractère **Chr(0)** alias **vbNullChar**. Vous pouvez utiliser la fonction ci-dessous pour accéder à la définition de la contrainte passée en paramètre :

```
Private Function GetRuleText(pRule As String) As String
On Error GoTo gestion_erreurs

Dim lCsts() As String
Dim lCpt As Long

lCsts = Split(Me.Recordset.Properties("CheckConstraints").Value, vbNullChar)
For lCpt = LBound(lCsts) To UBound(lCsts)
    If lCsts(lCpt) = pRule Then
        GetRuleText = lCsts(lCpt + 1)
    End If
Next
Exit Function
gestion_erreurs:
GetRuleText = ""
End Function
```



Ce qui donnerait dans la procédure vue plus haut :

```
    ' Construit le message personnalisé
    lNewMessage = "La règle " & lRule & " n'est pas validée :"
    lNewMessage = lNewMessage & vbCrLf & "Rappel de la règle : " & GetRuleText(lRule)
```

## IX - Conclusion

La mise en place de telles contraintes est certes un peu fastidieuse car éloignée de tout assistant graphique mais elle apporte un véritable plus en terme de cohérence et d'intégrité de vos données. En outre, en étant ainsi gérée directement par le moteur de base de données, vous avez la garantie qu'aucun programme tiers ne viendra à l'encontre des règles de gestion que vous avez définies. Ceci n'est vraiment pas négligeable quand on sait à quel point l'interconnexion des applications Office est mise en jeu dans les développements professionnels.

Car si à l'heure actuelle votre code VBA présent dans vos formulaires suffit à gérer les contraintes que vous avez énoncées, qu'en sera-t-il demain pour un autre programme réalisé par un autre prestataire qui attaquera directement votre base de données ?

J'adresse un remerciement tout particulier à Thierry Gasperment pour son astuce concernant la capture de l'erreur au sein d'un formulaire.