Migration des applications de DAO vers ADO

DAO n'est pas mort

Au vu des qualités de la technologie ADO, Microsoft recommande de convertir votre code DAO en code ADO. Ma recommandation est: **pas si vite!**

Il est vrai que la technologie ADO offre des améliorations pour le traitement des données, notamment du côté serveur. Toutefois elles ne s'appliquent pas nécessairement à votre situation si vous travaillez uniquement sur des bases de données gérées par Access et le Jet Engine. Même si vous écrivez des applications sous Jet 4.0 / Access 2000, vous avez encore besoin de la technologie DAO. Voici pourquoi.

DAO a été conçu et écrit en étroite liaison avec le moteur Jet. Il fournit des performances supérieures pour le traitement des tables Access / Jet. Si vous devez travailler avec plusieurs bases de données, vous pouvez utiliser DAO pour lancer une transaction en utilisant l'objet *Workspace* de Jet, ouvrir les bases de données et soit commit ou rollback pour terminer la transaction. Sous ADO l'objet *Workspace* n'existe pas, il est remplacé par un objet *Connection*, qui autorise l'accès à une seule base de données à la fois.

Une autre faiblesse de la technologie ADO est le verrouillage des enregistrements en cas de mise à jour simultanée. En effet, la propriété *LockType* ne permet pas d'appliquer le verrouillage général (pessimiste) sur une table Access en utilisant la constante intrinsèque *adLockPessimistic*, si vous utilisez aussi la syntaxe *Project.CurrentConnection*. Le verrou est simplement appliqué à la connexion active, et non pas aux autres connexions à la même table. Sous DAO on peut régler la propriété *LockEdit* sur *dbPessimistic* (valeur par défaut) et verrouiller les données pour tous les autres usagers jusqu'à la réalisation d'une modification. Si vous désirez empêcher l'accès des usagers à une table en lecture et en écriture, vous pouvez utiliser les constantes DAO *dbDenyRead* et *dbDenyWrite*. Il n'y a rien d'équivalent sous ADO.

Dans le cadre de la réplication avec Jet Replication Objects (JRO), extension à ADO disponible sous Jet 4.0, il manque aussi quelques possibilités que l'on trouve sous DAO. Par exemple, on ne peut pas créer un replica qui empêche les suppressions. On peut toutefois y parvenir via les menus Access 2000 en passant la valeur 4 à la méthode *MakeReplica* de DAO pour créer un replica bloquant les suppressions. De même JRO ne permet pas de régler la propriété *ReplicateProject* d'une base de données. En fait, la collection

Properties et la méthode *CreateProperty* ne sont pas disponibles sous ADO, JRO ou ADOX. A nouveau on doit utiliser DAO.

ADOX, soit l'ensemble des extensions ADO pour Jet 4.0 est déficient par rapport à DAO, notamment en matière de sécurité. Par exemple la méthode ADOX de création d'un nouvel usager ne permet pas de déterminer un PID (personal identifier), mais seulement le nom de l'usager et son mot de passe. La méthode génère un PID de manière aléatoire, auquel on ne peut pas accéder et par suite qu'on ne peut pas documenter. Or il est indispensable de documenter la sécurité dans une application critique pour une organisation! A nouveau il faut passer par DAO pour créer de nouveaux utilisateurs.

ADOX ne possède pas de propriété pour vérifier les permissions explicites et implicites (héritées) d'un usager sur un objet. Sous DAO on utilise la propriété *AllPermissions*. Sous ADOX il faut alors écrire du code pour vérifier si la permission existe dans tous les groupes auxquels un usager appartient.. Il est certain que ce code ne sera pas aussi efficace que celui de AllPermissions.

Sous ADOX il n'est pas possible d'attribuer ou de retirer les permissions sur les formulaires, les états, les macros et les modules. Microsoft a publié une procédure sur son site web mais elle ne fonctionne pas! Le seul recours est d'utiliser DAO. Cette technologie reste la plus conviviale pour fixer des liaisons entre tables, pour créer des indexes et pour la gestion des propriétés d'une base de données.

Ces exemples montrent bien qu'on ne peut pas renoncer à DAO tout de go. On peut toujours tirer parti des avantages des deux technologies en référençant les deux bibliothèques. En effet, si depuis Access on accède à des données externes gérées par exemple sous SQL Server ou Oracle, il est plus performant d'utiliser ADO. Comme le montre, plus avant, l'emploi de l'objet *Stream* pour **extraire / transmettre des données binaires depuis / vers une bdd SQL Server**.

A cet effet, je vous propose de consulter la page des astuces de migration entre les deux technologies, dont l'objectif est de faciliter une transition, toujours plus délicate qu'il n'y paraît de prime abord si on se réfère aux affirmations de Microsoft. En effet ces deux technologies présentent aussi d'importantes différences syntaxiques pour la réalisation d'une même tâche.

Résumé des problèmes de compatibilité.

Performance

- Si vous faites des mises à jour en traitement par lot en fin de journée. ADO nécessite environ 5 fois plus de temps.
- ADO présente une dégradation de 30 - 70% sur des requêtes utilisant des tables comportant quelques dizaines de rubriques. En effet, quand DAO ouvre un recordset, il n'y a pas besoin de lire les informations de schéma (noms de

rubriques et attributs) avant une demande explicite. Sous ADO ces informations sont lues systématiquement au début du processus d'ouverture. Ceci explique la lenteur des requêtes SQL accédant à toutes les rubriques des tables sous-jacentes. Il faut alors éviter dans la mesure du possible les SELECT *.

Emploi des ressources

- DAO ouvre uniquement une seule session Jet. Par contre ADO ouvre une session Jet par connexion. Ainsi, on arrive rapidement à épuiser le quota des sessions que Jet peut gérer.
- Des sessions Jet multiples présentent un délai d'attente de 5 secondes pour s'actualiser les unes les autres.

Curseurs Client et Serveur

- Sous ADO on peut utiliser des curseurs côté Client pour extraire des données et les stocker localement. Ceci offre de nouvelles fonctionnalités tel que la sauvegarde et la déconnexion des recordsets ou la mise à jour en traitement par lots.
- Ceci signifie aussi que les données copiées localement ne refléteront pas les modifications et les ajouts réalisés par les autres usagers tant qu'on n'a pas réactualisé les données. De plus, la copie des enregistrements sur un fichier local temporaire entraîne une dégradation des performances.
- Par contre si on utilise les curseurs côté Serveur, on laisse le moteur Jet gérer les enregistrements plus rapidement et présenter des données actualisées en permanance. Toutefois, on ne disposera pas des nouvelles fonctionalités offertes sous ADO.

Volume des applications

Quand on distribue une application DAO, le nombre et la taille des fichiers supportant la technologie DAO sont réduits. Par contre, si on développe sous ADO, on est obligé de distribuer plusieurs bibliothèques volumineuses et des convertisseurs de données source non utilisés par l'application.

Recherche des enregistrements

Les méthodes Find de DAO ne requièrent pas de fonctionnalité spécifique du moteur Jet. Par contre ADO doit implémenter des méthodes Find au niveau du moteur. Il en résulte une recherche plus lente et moins souple.

Sécurité

- DAO offre un modèle complet de gestion de la sécurité au niveau utilisateur.
- Sous ADO on ne peut pas accéder au PID des usagers. Cela signifie que si, par malheur, on perd les informations stockées dans le fichier *system.mdw*, on est perdu, car il n'y a pas moyen de recréer les groupes et les utilisateurs sans connaître le contenu de leur PID.

- De plus, ADO ne parvient pas à gérer la sécurité des formulaires, des états, des macros et des modules.

Langage de définition des données

ADO tente de supporter les mêmes fonctionnalités DDL que DAO pour la création de nouvelles tables, des indexes, des liaisons, etc. Leur implémentation rend les opérations bien plus complexes que sous DAO. Pour s'en convaincre, il suffit de tenter la création d'une rubrique indexée avec un ordonnancement descendant.

En conclusion de cette première comparaison des deux technologies d'accès aux données, je conseille de garder l'essentiel du code DAO et de recourir à ADO quand on désire exploiter les nouvelles fonctionnalités offertes par Jet 4.0 ou quand on veut bénéficier des possibilités fournies par ADO qui n'existent pas sous Jet.

Haut du document Trucs et Astuces MS-Access

Migration de DAO vers ADO

Tout d'abord, on va faire sous ADO ce que nous savons faire sous DAO, après avoir référencé la Microsoft Active X Data Object Library x.y.

Ouverture d'une base de données
L'approche ADO est similaire à celle de DAO, mais on ouvre un nouvel objet *Connection*.

```
Sub OuvrirBdd()

     Dim cnn As New ADODB Connection
     ' Il faut indiquer la version du moteur de bdd (provider) et
     ' la source des données (le fichier .mdb).
     cnn.Open _
     "Provider=Microsoft.Jet.OLEDB.4.0;Data
     Source=D:\Test.mdb;"
     ' Fermer la connexion équivalent de fermer la bdd sous DAO.
     cnn.Close
End Sub
```

Ouvrir une bdd en mode lecture seule
On doit régler la propriété Mode de l'objet Connection.

```
Sub OuvrirBddLectureSeule()

     Dim cnn As New ADODB Connection
     cnn.Mode = adModeRead
     cnn.Open _
```

```
     "Provider=Microsoft.Jet.OLEDB.4.0;Data
     Source=D:\Test.mdb;"
     cnn.Close
End Sub
```

Ouvrir une bdd non gérée par le moteur Jet
Sous DAO on peut utiliser *OpenDatabase* pour ouvrir un fichier ISAM non géré par Jet, par exemple une feuille Excel. Sous ADO il faut utiliser l'argument *Extended Properties* dans la chaîne de connexion pour spécifier le programme de pilotage.

```
Sub OuvrirBddISAM()

     Dim cnn As New ADODB Connection
     cnn.Open _
     "Provider=Microsoft.Jet.OLEDB.4.0;Data
     Source=D:\Taxe.xls;" &_ "Extended Properties=Excel
     8.0;"
     cnn.Close
End Sub
```

Ouvrir un recordset en lecture avant
Dans la méthode Open on devra spécifier les arguments *CursorType* et *LockType* pour limiter le traitement à la lecture avant.

```
Sub OuvrirRecordsetLectureAvant()

     Dim cnn As New ADODB Connection
     Dim rs As New ADODB.Recordset
     cnn.Open _
     "Provider=Microsoft.Jet.OLEDB.4.0;Data
     Source=D:\Test.mdb;"
     rs.Open _
     "SELECT * FROM tblClients WHERE Devise = 'CHF'", _
     cnn, adOpenForwardOnly, adLockReadOnly
     rs.Close
     cnn.Close
End Sub
```

Ouvrir un recordset modifiable fondé sur une requête virtuelle
Sous ADO il n'y a pas de méthode *Edit* car on est automatiquement dans ce mode. De même il n'est pas nécessaire de recourir à la méthode *Update* pour enregistrer une modification avant de passer à l'enregistrement suivant, car ADO réalise cela pour vous. On renseigne l'argument *CursorType* avec *adOpenKeyset* pour autoriser la mise à jour. Mais attention, contrairement à DAO, les modifications pendantes sont enregistrées automatiquement si vous quittez un enregistrement. Pour annuler une mise à jour, il faut utiliser la méthode *CancelUpdate*.

```
Sub ModifierRecordset()
```

```
    Dim cnn As New ADODB Connection
    Dim rs As New ADODB.Recordset
    cnn.Open _
    "Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=D:\Test.mdb;"
    rs.Open _
    "SELECT * FROM tblClients WHERE CodeClient = 'BCV'",
    _
    cnn, adOpenKeyset, adLockOptimistic
    rs.Fields("NomContact").Value = "Yves Christen"
    rs.Close
    cnn.Close
End Sub
```

Créer un recordset avec une instruction SQL et le lister
Pour lister le Recordset, on utilisera avantageusement la méthode GetString.

```
Sub ListerRecordsetSQL()

    Dim cnn As New ADODB Connection
    Dim rs As New ADODB.Recordset
    cnn.Open _
    "Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=D:\Test.mdb;"
    rs.Open _
    "SELECT * FROM tblClients", , adOpenForwardOnly,
    adLockReadOnly, adCmdText
    Debug.Print rs.GetString
    rs.Close
    cnn.Close
End Sub
```

La méthode **GetString** permet de remplacer une boucle répétitive.
Elle comporte cinq arguments. Le premier est la constante adClipString (seul choix possible), qui spécifie le format du recordset (chaîne de car.) Le second argument indique le nombre d'enregistrements à lister (Par défaut: tous). Le 3ème argument indique le délimiteur de rubriques à l'intérieur de l'enregistrement (Par défaut: tabulateur).
Le 4ème argument est le délimiteur d'enregistrements (Par défaut: retour charriot). Le 5ème argument est une expression de représentation des valeurs nulles (Par défaut: "").

Exemple: `Debug.Print rsClients.GetString(adClipString, 10, "; ")`

Ouverture d'un recordset fondé sur une requête stockée
On va exécuter la requête *Produits A Prix Special* de la Bdd et l'exploiter en lecture avant. Sous ADO, les noms d'objets contenant des espaces doivent être encadrés par les signes **[** et **]**.

```
Sub OuvrirReqBdd()

     Dim cnn As New ADODB Connection
     Dim rs As New ADODB.Recordset
     cnn.Open _
     "Provider=Microsoft.Jet.OLEDB.4.0;Data
     Source=D:\Test.mdb;"
     rs.Open "[Produits A Prix Special]", _
     cnn, adOpenForwardOnly, adLockReadOnly
     rs.Close
     cnn.Close
End Sub
```

Ouvrir une requête paramétrable

Au lieu d'utiliser la collection Parameters comme sous DAO, on utilise l'objet *Command* et la méthode *CreateParameter*.

```
Sub OuvrirReqParametrable)()
Dim cnn As ADODB Connection
Dim rs As New ADODB.Recordset
Dim cmd As ADODB.Command
Set cnn = New ADODB.Connection
With cnn

     .Provider = "Microsoft.Jet.OLEDB.4.0"
     .ConnectionString = "Data Source=" App.Path &
     "\Essai.mdb"
     .Mode = adModeReadWrite
     .Open
End With
Set cmd = New ADODB.Command
With cmd
     .CommandText = "ExempleParam"
     .CommandType = adCmdStoreProc
     .ActiveConnection = cnn
     .Parameters.Append.CreateParameter("CodeClient", _
     adWChar, adParamInput, 5, "Moilneu")
End With
Set rs = cmd.Execute
Do Until rs.EOF
     Debug.Print rs(0)
     rs.MoveNext
Loop
rs.Close
cnn.Close
Set rs = Nothing
Set cmd = Nothing
Set cnn = Nothing
End Sub
```

Déterminer le nombre d'enregistrements affectés par une requête action
Sous DAO on dispose de la propriété de bdd RecordsAffected. Sous ADO on doit créer une variable de type entier long, qui sera passée à la méthode Execute, et sera remplie par ADO. Dans cet exemple, le paramètre Option de la méthode Execute est réglé sur *adExecuteNoRecords* pour indiquer qu'il n'y a pas de changement dans le nombre d'enregistrements.

```
Sub ExecuterReqAction()

     Dim cnn As ADODB.Connection
     Dim lngAffectes As Long
     Dim strSQL As String
     cnn.Open _
     "Provider=Microsoft.Jet.OLEDB.4.0;Data
     Source=D:\Test.mdb;"
     strSQL = "UPDATE tblClients SET Pays = 'Suisse';"
     cnn.Execute strSQL, lngAffectes, adExecuteNoRecords
     Debug.Print "Nbre d'enregistrements affectés: " &
     lngAffectes
     cnn.Close
End Sub
```

Recherche séquentielle d'un enregistrement
Bien que disposant aussi de la méthode Find, la technologie ADO fonctionne différemment pour se déplacer dans un ensemble d'enregistrements. Au lieu des méthodes *FindFirst* et *FindNext* et de la propriété *NoMatch*, on utilise les paramètres *SkipRecords* et *SearchDirection* de la méthode Find.

On utilise la méthode Find pour trouver la concordance suivante. En réglant **SkipRecords** sur **1** on demande à ADO de quitter l'enregistrement actif et de rechercher le suivant. Le paramètre SearchDirection permet de préciser le sens de la recherche: avant ou arrière.

```
Sub RechercheSeqEnreg()
Dim cnn As New ADODB Connection
Dim rs As New ADODB.Recordset
cnn.Open _
"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\Test.mdb;"
rs.Open "tblClients", _
cnn, adOpenKeyset, adLockOptimistic
' Trouve le premier enregistrement concordant
rs.Find "Pays='Suisse'"
' Affiche la liste des clients suisses
Do Until rs.EOF

     Debug.Print rs.Fields("CodeClient").Value
     ' Recherche l'enregistrement concordant suivant en lecture avant
     rs.Find "Pays='Suisse'", SkipRecords:=1, _
     SearchDirection:=adSearchForward
```

```
Loop
rs.Close
cnn.Close
End Sub
```

Recherche directe d'un enregistrement

Sous ADO il faut non seulement ouvrir le recordset et spécifier l'index, mais en plus il faut indiquer la position du curseur. Ceci est nécessaire car la technologie ADO ne supporte pas directement les indexes. Elle passe une demande de ce type de recherche au moteur du SGBD placé sur le serveur. D'autre part, quand on veut effectuer une recherche multi-clés, il faut créer un tableau de valeurs de type Variant. Dans l'exemple on veut rechercher un enregistrement de la table DetailCommandes identifié par le NoCommande 1234 et le CodeProduit 55. Sous ADO la méthode Seek ne fonctionne pas du côté Client (*adUseClient*), mais elle offre des options supplémentaires telle que la limitation à la première concordance (*adSeekFirstEQ*).

```
Sub RechercheDirecteEnreg()
Dim cnn As New ADODB Connection
Dim rs As New ADODB.Recordset
cnn.Open _
"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\Test.mdb;"
rs.Index =PrimaryKey"
rs.CursorLocation = adUseServer
' Le paramètre adCmdTableDirect indique le type de recordset qui supporte
l'usage
' de la propriété Index
rs.Open "tblDetailCommandes", cnn, adOpenKeyset, _
adLockOptimistic, adCmdTableDirect
rs.Seek Array(1234, 55), adSeekFirstEQ
' Si trouvé, affiche la quantité commandée
If Not rs.EOF Then

     Debug.Print rs.Fields("QtéCdée").Value
End If
rs.Close
cnn.Close
End Sub
```

Filtrer des enregistrements

Sous ADO, dès que la propriété *Filter* est renseignée le recordset ouvert est repeuplé automatiquement, tandis que sous DAO il faut créer un second objet recordset.

```
Sub FiltrerEnreg()
Dim cnn As New ADODB Connection
Dim rs As New ADODB.Recordset
cnn.Open _
"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\Test.mdb;"
```

```
rs.Open "tblClients", _
cnn, adOpenKeyset, adLockOptimistic
rst.Filter = "Pays='Suisse' And Email<>Null"
Do Until rs.EOF

     Debug.Print rs.Fields("CodeClient").Value
     rs.MoveNext
Loop
rs.Close
cnn.Close
End Sub
```

Trier des enregistrements

Sous ADO dès que la propriété *Sort* est renseignée le recordset ouvert est trié automatiquement, tandis que sous ADO il faut créer un second objet recordset.

```
Sub TrierEnreg()
Dim cnn As New ADODB Connection
Dim rs As New ADODB.Recordset
cnn.Open _
"Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\Test.mdb;"
rs.Open "tblClients", _
cnn, adOpenKeyset, adLockOptimistic
rst.Sort = "TitreContact"
Do Until rs.EOF

     Debug.Print rs.Fields("TitreContact").Value
     rs.MoveNext
Loop
rs.Close
cnn.Close
End Sub
```

Exécuter des transactions pour garantir la cohérence des MAJ

Sous DAO on gère une transaction au niveau de la session, tandis que sous ADO on la gère au niveau de l'objet Connection.

```
Sub ExecTransaction()

     Dim cnn As New ADODB Connection
     cnn.Open _
     "Provider=Microsoft.Jet.OLEDB.4.0;Data
     Source=D:\Test.mdb;"
     cnn.BeginTrans
     cnn.Execute "UPDATE tblClients SET
     TypeClient='Etranger'"
     cnn.CommitTrans
     cnn.Close
End Sub
```

Programmer avec ADOX

Pour programmer le DDL et les tâches du moteur Jet, on recourt à la bibliothèque **ADOX** (Microsoft ADO Ext. x.y) contenant les programmes de gestion des tâches système. Il s'agit en fait d'une extension de la bibliothèque ADO.

Créer une base de données
Sous DAO on peut accéder directement à la bdd créée car la méthode CreateDatabase fournit un descripteur. Sous ADO il faut d'abord l'ouvrir. On devra aussi référencer la bibliothèque **ADOX**. Cette bibliothèque comporte un objet *Catalog* qui est l'équivalent de l'objet Database sous DAO.

```
Sub CreerBdd()

     Dim cat As New ADOX.Catalog
     cat.Create _
     "Provider=Microsoft.Jet.OLEDB.4.0;Data
     Source=D:\New.mdb;"
     Set cat = Nothing
End Sub
```

Optimiser le fonctionnement du moteur du SGBD
On veut par exemple modifier le paramètre *PageTimeOut* qui définit le délai d'attente, en millisecondes, avant que le moteur ne vérifie si d'autres usagers ont apporté des modifications aux enregistrements de la page.

Au lieu d'utiliser la méthode *SetOption* spécifique au Jet Engine, ADO fournit un mécanisme générique pour passer les informations aux divers moteurs de bdd.

```
Sub OptionJet()

     Dim cnn As New ADODB.Connection
     cnn.Provider = "Microsoft.Jet.OLEDB.4.0"
     cnn.Properties("Jet OLEDB:Page Timeout") = 4000
     cnn.Open "D:\Test.mdb;"
     cnn.Close
End Sub
```

Documenter la structure de la bdd
Par exemple on veut documenter les tables. Sous ADO, la collection Tables contient toutes les tables et des vues. Il faut alors utiliser la propriété *Type* de l'objet Table pour obtenir des informations sur la nature précise des objets et des détails supplémentaires..

```
Sub ListerTables()
Dim cat As New ADOX.Catalog
```

```
Dim tbl As ADOX.Table
Dim strTmp As String
' Ouvrir un objet Catalog pour la bdd
cat.ActiveConnection = _
"Provider=Microsoft.Jet.OLEDB.4.0;" & _
"Data Source=D:\Test.mdb;"
For Each tbl In cat.Tables

     strTmp = tbl.Name
     Select Case tbl.Type
          Case "LINK"
               strTmp = strTmp & "(Liée " & _
               tbl.Properties("Jet OLEDB:Link
               Datasource") & _
               ", Table=" & tbl.Properties("Jet
               OLEDB:Remote _
               Table Name") & ")"
          Case "ACCESS TABLE"
               strTmp = strTmp & " (Table système
               Access)"
          Case "TABLE"
               strTmp = strTmp & " (Table)"
          Case "SYSTEM TABLE"
               strTmp = strTmp & " (Table système Jet)"
          Case "VIEW"
               strTmp = strTmp & " (Vue utilisateur)"
          Case Else
               strTmp = strTmp & " (type inconnu:" &
               tbl.Type & ")"
     End Select
     Debug.Print strTmp
Next
Set cat = Nothing
End Sub
```

Si on veut lister les requêtes stockées, on recourt à la collection **Views**.

```
Sub ListerRequetes()
Dim cat As New ADOX.Catalog
Dim vue As ADOX.View
cat.ActiveConnection = _
"Provider=Microsoft.Jet.OLEDB.4.0;" & _
"Data Source=D:\Test.mdb;"
For Each vue in cat.Views

     Debug.Print vue.Name
Next
Set cat = Nothing
End Sub
```

Créer une table
Au lieu de créer des objets Field et de les ajouter à la définition de la table, on utilise la méthode **Append** de l'objet **Columns**.

```
Sub CreerTable()
Dim cat As New ADOX.Catalog
Dim tbl As New ADOX.Table
Dim col As New ADOX.Column
cat.ActiveConnection = _
"Provider=Microsoft.Jet.OLEDB.4.0;" & _
"Data Source=D:\Test.mdb;"
With tbl

     .Name = "NouvelleTable"
     .Columns.Append "Integer", adInteger
     .Columns.Append "UnsignedTinyInt", adUnsignedTinyInt
     .Columns.Append "Single", adSingle
     .Columns.Append "Double", adDouble
     .Columns.Append "Currency*, adCurrency
     .Columns.Append "Boolean", adBoolean
     .Columns.Append "GUID", adGUID
     .Columns.Append "Date", adDate
     .Columns.Append "Wchar", adWChar, 30
     .Columns.Append "Varwchar", adVarWChar
     .Columns.Append "LongVarWChar", adLongVarWChar
     .Columns.Append "Binary", adBinary, 100
     .Columns.Append "VarBinary", adVarBinary
     .Columns.Append "LongVarBinary", adLongVarBinary
     .Columns.Append ""CounterField", adInteger
     ' On veut un compteur autonumérique débutant à 100 avec un
     incrément de 10
     Set col = tbl.Columns("CounterField")
     With col
          Set .ParentCatalog = cat
          .Properties("AutoIncrement") = True
          .Properties("Seed") = CLng(100)
          .Properties("Increment") = CLng(10)
     End With
End With
cat.Tables.Append tbl
Set cat = Nothing
End Sub
```

Transfert de données XML
L'objet Stream de la technologie ADO possède différentes méthodes et propriétés utilisées en fonction des types de données traitées. Ainsi, pour lire du texte on utilise la méthode *ReadText* et pour lire du binaire on utilise la méthode *Read*.

L'exemple suivant montre comment on prépare un objet Stream à transférer des données XML.

```
Dim strm As ADODB.Stream
Set strm = New ADODB.Stream
With strm

     .LineSeparator = adCRLF
     .Mode = adModeRead
     .Type = adTypeText
     .Open
End With
```

L'objet *Command* de la technologie ADO offre un moyen de placer des données dans un objet Stream. A cet effet, l'objet Command dispose de deux propriétés. La propriété *Output Stream* indique l'objet Stream dans lequel il faut placer les données. La propriété *XML Root* indique à ADO le nom du node racine du document XML résultant. On accède à ces propriétés de l'objet Command comme suit:

```
cmd.Properties("Output Stream") Value = strm
cmd.Properties("XML Root") Value = "data"
```

Après avoir renseigné ces propriétés et les propriétés plus connues telles que CommandType et CommandText, on utilise la commande adExecuteStream pour demander à ADO d'utiliser le pipeline. Alors ADO place les données XML indiquées dans l'objet Stream spécifié.

A cet instant les données sont encore en format binaire. Pour les convertir en format texte, on utilise la méthode ReadText de l'objet Stream. Cette méthode possède un argument qui indique la quantité de texte que vous désirez extraire. Les valeurs permises de cet argument sont: *adReadAll*, *adReadLine* ou le nombre de caractères à extraire.

Quand on lit des lignes individuelles ou un certain nombre de caractères, par exemple 1000, on va tester la fin du fichier avec la propriété EOS (End Of Stream) comme suit:

```
Do Until strm.EOS

     strXML = strXML & strm.ReadText(1000)
Loop
```

ADO va boucler automatiquement à travers le fichier à chaque appel de la méthode ReadText.

Pour plus de détails, il faut étudier les fichiers d'aide ADO livrés avec Access et Office 2000 / 2002.

Haut du document Trucs et Astuces MS-Access

Programmer la sécurité Jet Engine sous ADO

Ouverture d'une bdd protégée par un mot de passe
On va passer le mot de passe lors de la connexion selon un format spécifique au moteur du SGBD accédé.

```
Sub OuvrirBddMotPasse()
Dim cnn As New ADODB.Connection
cnn.Open _

     "Provider=Microsoft.Jet.OLEDB.4.0;" & _
     "Data Source=D:\Test.mdb;" & _
     "Jet OLEDB:Database Password=Saturne95;"
cnn.Close
End Sub
```

Créer un mot de passe de Bdd
Sous DAO on dispose de la méthode *NewPassword*. Sous ADO, il faut utiliser la utiliser la méthode *CompactDatabase* de **JRO** et sa propriété *password*. Cela signifie qu'on ne peut pas assigner un mot de passe de bdd sans compacter la bdd. Ceci est donc relativement complexe. Voici tout de même un exemple complet des opérations à entreprendre sous ADO pour protéger une bdd avec un mot de passe.

```
Sub CreerMotpasseBdd()
Dim JROJet As New JROP.JetEngine
Dim strBdd As String, strNew As String
Dim strMotPasseExistant As String
strBdd = "D:\Test.mdb"
strNew = "D:\New.mdb"
strMotPasseExistant = "Neptune02"
If Dir(strNew) <> "" Then

     Kill strNew
End If
On Error Resume Next
JROJet.CompactDatabase _
     "Data Source=" & strBdd & ";" & _
     "Jet OLEDB:Database Password=strMotPasseExistant, _
     "Data Source=" & strNew & ";" & _
     "Jet OLEDB:Database Password=NouveauMotPasse"
If Err.Number <> 0 Then
     MsgBox "Erreur de compactage: " & Err.Description
Else
     Kill strBdd
     Name strNew As strBdd
End If
End Sub
```

Changer un mot de passe utilisateur
La bibliothèque ADOX offre la méthode *ChangePassword* de l'objet *Catalog*. On

l'utilise pour changer un mot de passe au niveau groupe ou au niveau usager. Sous ADO on doit ouvrir une connexion vers la base de données système d'Access.

```
Sub ChangerMotPasse()
Dim cat As New ADOX.Catalog
cat.ActiveConnection = _

     "Provider=Microsoft.Jet.OLEDB.4.0;" & _
     "Data Source=D:\Test.mdb;" & _
     "Jet OLEDB:System Database=D:\system.mdw"
cat.Users("Admin").ChangePassword "", "Confidentiel"
set cat = Nothing
End Sub
```

Créer un nouvel utilisateur

Cette opération doit être réalisée sous DAO, car comme déjà relevé on ne peut pas documenter le PID (No personnel d'identification) sous ADO, ce qui est vital si on doit recréer le système de sécurité. Il ne faut donc pas utiliser ADO pour **créer** des utilisateurs. On rappellera ici comment procéder sous DAO.

```
Sub CreerUsager()
Dim wks As DAO.Workspace, usr As DAO.User
DBEngine.SystemDB = C:\system.mdw
'On suppose qu'il n'y a pas de mot de passe pour Admin
Set wks = DBEngine.CreateWorkspace("", "Admin", "")
On crée l'usager Einstein avec un pid de EMC2 et le mot de passe 1879
Set usr = wks.CreateUser("Einstein", "EMC2", "1879")
wks.Users.Append usr
wks.Close
End Sub
```

Ajouter un utilisateur à un groupe

L'objet *Catalog* de la bibliothèque ADOX possède aussi la collection *Users* ce qui permet d'ajouter un utilisateur à un groupe existant. On va ajouter l'utilisateur Einstein au groupe Recherche.

```
Sub AjoutUsagerGroupe()
Dim cat As New ADOX.Catalog
cat.ActiveConnection = "Provider=Microsoft.Jet.OLEDB.4.0;" & _

     "Data Source=D:\Test.mdb;" & _
     "User Id=Admin;" & _
     "Password=password;" & _
     "Jet OLEDB:System database=C:\system.mdw"
cat.Users("Einstein").Groups.Append "Recherche"
Set cat = Nothing
End Sub
```

Créer un nouveau groupe
Il s'agit d'ajouter un nouvel objet à la collection *Groups* de l'objet *Catalog*. On va créer le groupe Direction.

```
Sub CreerGroupe()
Dim cat As New ADOX.Catalog
cat.ActiveConnection = "Provider=Microsoft.Jet.OLEDB.4.0;" & _

     "Data Source=D:\Test.mdb;" & _
     "User Id=Admin;" & _
     "Password=password;" & _
     "Jet OLEDB:System database=C:\system.mdw"
cat.Groups.Append "Direction"
Set cat = Nothing
End Sub
```

Accorder des permissions d'accès à un objet de la bdd
Pour l'instant il est déconseillé d'utiliser la technologie ADO à cet effet, car elle est embryonnaire dans ce domaine. On continue donc d'utiliser DAO pour assurer la confidentialité en réglant la propriété *Permissions* de l'objet *Document* de la collection *Containers* appropriée.

```
Sub AccorderPermissions()
Dim Bdd As DAO.Database, wks As DAO.Workspace
Dim doc As DAO.Document
DBEngine.SystemDB = "C:\system.mdw"
Set wks = DBEngine.CreateWorkspace("", "Admin", "")
Set Bdd = wks.OpenDatabase(D:\Test.mdb")
' Accorder des permissions à l'usager Einstein sur tblEnergie
Set doc = Bdd.Containers("Tables").Documents("tblEnergie")

     doc.UserName = "Einstein"
     doc.Permissions = _
          dbSecInsertData And dbSecReplaceData And
          dbSecDeleteData
Bdd.Close
wks.Close
End Sub
```

Encrypter la bdd avec le service intégré
Ce service de cryptage permet simplement d'empêcher l'ouverture de votre bdd avec un éditeur de fichiers au niveau du DOS. Le moteur du SGBD supporte le cryptage en utilisant des options de la méthode *CompactDabase*. Sous ADO on doit utiliser la méthode CompactDatabase de JRO.

```
Sub CrypterBdd()
Dim JROJet As New JRO.JetEngine
JROJet.CompactDatabase _
```

```
     "Data Source=D:\Test.mdb;", _
     "Data Source=D:\TestCrypt.mdb;" & _
     "Jet OLEDB:Encrypt Database=True
End Sub
SubDecrypterBdd()
Dim JROJet As New JRO.JetEngine
JROJet.CompactDatabase _
     "Data Source=D:\TestCrypt.mdb;", _
     "Data Source=D:\Test.mdb;" & _
     "Jet OLEDB:Encrypt Database=False
End Sub
```

Pour plus de détails, il faut étudier les fichiers d'aide ADO livrés avec Access et Office 2000 / 2002.