5 ACCES AUX DONNEES

VB dispose de nombreux moyens d'accéder aux données. Nous allons voir essentiellement comment exploiter une base Access.

Il s'agit d'une des fonctionnalités les plus exploitées en VB. De nombreux objets sont mis à disposition pour manipuler la structure de bases de données, ainsi que les données contenues par ces bases.

Par ailleurs, VB permet de passer des ordres SQL aux moteurs de bases de données sous forme de chaînes de caractères. La dernière version normalisée de SQL remonte à 1992 avec l'ANSI92. Par la suite les éditeurs ont introduit quelques variantes, mais la version standard est reconnue par la quasi-totalité des moteurs de bases de données.

5.1 La base de registre

La base de registre est un élément essentiel du système d'exploitation. Elle contient le paramétrage des applications et de Windows. On trouve les ID des DLL, des OCX, les associations de fichiers aux applications, les paramètres de connexion, les profils utilisateurs…

On peut l'utiliser en VB pour sauvegarder des paramètres dans une clé spécifique (*VB and VBA Programs setting de CurrentUser*). Cette clé est la racine de tous les paramètres accessibles directement en VB ou VBA. On dispose de quatre fonctions qui utilisent des chaînes ou tableaux de chaînes.

GetSetting(appname, section, key[, default])

Cette fonction lit le registre et renvoie la valeur de la clé sélectionnée.

appname Expression de chaîne contenant le nom de l'application ou du projet dont vous extrayez une valeur de clé.

section Expression de chaîne contenant le nom de la section où se trouve la valeur de clé.

key Expression de chaîne contenant le nom de la valeur de clé à renvoyer.

default Facultatif. Expression contenant la valeur à renvoyer si la clé ne contient pas de valeur ou si elle n'existe pas. Si l'argument default est omis, une chaîne de longueur nulle ("") est retournée.

SaveSetting appname, section, key, value

Cette instruction écrit une valeur dans le registre. Si la clé n'existe pas, elle automatiquement créée.

appname Expression de chaîne contenant le nom de application ou du projet concerné.

section Expression de chaîne contenant le nom de la section dans laquelle la valeur de clé doit être enregistrée.

key Expression de chaîne contenant le nom de la valeur de clé à enregistrer.

value Expression contenant la valeur attribuée à key.

↳ DeleteSetting appname, section[, key]

Cette instruction permet de supprimer une clé ou une section du registre.

appname Expression de chaîne contenant le nom de l'application ou du projet auquel s'applique la valeur de section ou de clé.

section Expression de chaîne contenant le nom de la section dans laquelle la valeur de clé est supprimée. Si seuls les arguments appname et section sont indiqués, la section définie est supprimée en même temps que les valeurs de clé connexes.

key Facultatif. Expression de chaîne contenant le nom de la clé à supprimer.

↳ GetAllSettings(appname, section)

Cette fonction renvoie la liste des clés d'une section. Il s'agit d'un variant contenant un tableau de chaînes à deux dimensions. Une dimension stocke le nom des clés, l'autre les valeurs correspondantes.

appname Expression de chaîne contenant le nom de l'application ou du projet dont vous voulez extraire les valeurs de clé.

section Expression de chaîne contenant le nom de la section dont vous voulez extraire les valeurs de clé. La fonction GetAllSettings renvoie une valeur de type Variant contenant un tableau de chaînes à deux dimensions où sont stockées les clés de la section indiquée et leur valeur.

5.2 Accès aux fichiers

5.2.1 Accès aux fichiers texte

VB permet d'accéder aux fichiers texte en lecture et en écriture alternativement. Il n'est pas possible de lire et d'écrire simultanément le même fichier.

Pour ouvrir un fichier en lecture, on utilise l'instruction suivante :

```
Open pathname For mode [lock] As [#]filenumber
```

pathname Expression de chaîne indiquant un nom de fichier ; peut comprendre un nom de répertoire ou de dossier et un nom de lecteur (chemin complet).

mode Mot clé indiquant le mode d'ouverture du fichier : `Input`, `Output`. S'il n'est pas indiqué, le fichier est ouvert en mode Random (voir la section suivante).

lock Facultatif. Mot clé indiquant les opérations autorisées sur le fichier ouvert par d'autres processus : *Shared, Lock Read, Lock Write* et *Lock Read Write*.

Filenumber Numéro de fichier valide compris entre 1 et 511, inclus. Utilisez la fonction *FreeFile* pour obtenir le prochain numéro de fichier disponible.

Ex :

```
Open « C:\test\monfichier.txt » For Input As #1
```

On utilise ensuite le numéro du fichier (#1) pour y accéder à l'aide de commande de lecture.

L'instruction *Input #* permet de récupérer le texte écrit dans un fichier séquentiel à l'aide de l'instruction *Write #* (fichier ouvert en mode *Output*).

Ex :

```
Input #1, strVar1          '#1 est le numéro de fichier et strVar1 est une variable String dans laquelle est stocké le résultat de la lecture.
```

Une variante de cette instruction existe sous la forme *Line Input #*. Elle permet de lire un fichier texte ligne par ligne. Le caractère retour chariot marque la fin de chaque ligne, mais il n'est pas retourné par cette instruction. On utilise cette commande sur les fichiers écrits à l'aide de l'instruction *Print #*.

```
Ex : Line Input #1, strVar1
```

Nous avons donc vu qu'il y a deux instructions pour écrire dans des fichiers texte. Il faut que le fichier ait été ouvert en mode *Output*. Il s'agit de *Write #* et *Print #*, et elles s'utilisent comme suit :

```
Print #filenumber, [outputlist]     '[outputlist] correspond à une expression à inscrire dans le fichier.

Write #filenumber, [outputlist]
```

A la fin des manipulations de fichier, il faut les refermer. On utilise l'instruction *Close #* selon le modèle suivant :

```
Close [[#]filenumber] [, [#]filenumber] . . .
```

Si on utilise *Close* sans paramètre, tous les fichiers ouverts avec l'instruction *Open* seront fermés. Après fermeture, la référence au numéro de fichier est libérée.

Exemple :

```
Private Sub ReadFile()
Dim strFic As String
Dim strTmp As String

        'Ouverture du fichier en lecture
        Open « c:\test\msg.txt » For Input As #2

        Do Until EOF(2)
                Line Input #2, strTmp 'Récupère les lignes une à une
                 'Ajout à une var récupérant l'ensemble du fichier. Il faut insérer le retour chariot qui est omis par
                Line Input.
                strFic = strFic & vbCrLf & strTmp
        Loop
        'Fermeture du fichier
        Close #2

        'Ouverture du fichier en écriture
Open « c:\test\msg.txt » For Output As #2
'Ajoute une chaîne à l'ancien contenu
        Print #2, strFic & vbCrLf & « Et Voilà »
        Close #2

End Sub
```

5.2.2 *Accès aux fichiers binaires*

L'accès aux fichiers binaires se fait avec la même commande *Open* que précédemment. C'est le paramètre *Mode* qui précise le type d'accès. Il en existe plusieurs dans le cas des fichiers binaires. Vous trouverez le détail des différents modes dans l'aide sur la commande *Open*.

Les techniques de lecture et d'écriture sont également très proches. Les mots réservés changent. Reportez-vous à l'aide en ligne pour trouver tous les renseignements sur ces techniques.

5.3 Le langage SQL

Structured Query Language

Nous ne développerons pas ce chapitre, mais la connaissance de SQL est essentielle pour exploiter VB dans le cadre d'applications de gestion de données.

VB permet d'envoyer des ordres SQL aux moteurs de base de données et permet d'exécuter toutes les instructions de SQL. Ce langage, bien qu'il ne soit plus normalisé, est un standard de manipulation de données.

Pour accéder à une base de données, VB s'appuie sur trois bibliothèques :

- OLE DB assure la connexion ;
- ADO permet de manipuler les données ;
- ADOX établit un accès à la structure et à la sécurité.

Le modèle d'accès aux données précédent, DAO regroupait les objets de ADO et ADOX dans une même bibliothèque. La connexion était fréquemment assurée par ODBC.

5.4 Organisation de l'accès aux données stockées dans un SGBD

5.4.1 *Présentation*

VB peut exploiter plusieurs modèles objets pour accéder aux bases de données. Le choix se fait en fonction de la base utilisée (Access, SQL Server, Oracle…) et selon le contexte souhaité (client /serveur, mono-poste, n tiers…).

Le modèle objet ADO est le plus récent. Les MDAC (Microsoft Data Access Component) contiennent une série d'objets d'accès aux données, dont ADO et ADOX.

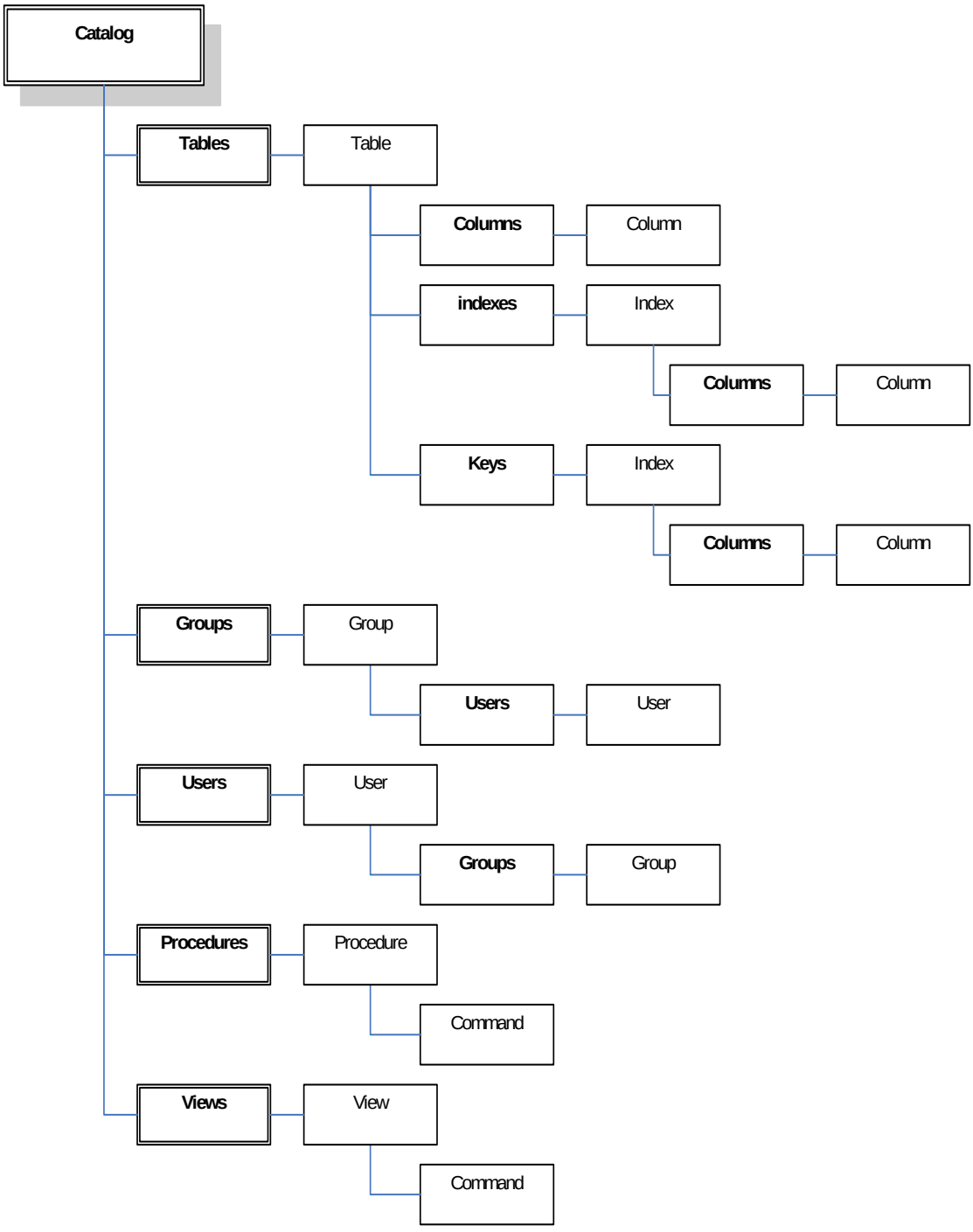
Nous allons nous intéresser à la manipulation des données plutôt qu'à la gestion de la structure. Dans la pratique, cette tâche est effectuée à l'aide d'outils livrés avec le SGBD.

Une base s'organise en deux sections essentielles qui sont le contenant qui correspond à la structure de la base, et le contenu correspondant aux données.

Les exemples et exercices peuvent être faits dans Access. Cet outil sert notamment à créer les bases.
L'aide en ligne et le site www.msdn.com sont des références importantes pour rentrer en détail dans les objets d'ADO et ADOX. Bien sûr, il existe d'autres références qui pourront vous aider.

5.4.2 Les objets de structures de base de données

Voici le modèle objets global d'ADOX. Vous trouverez des explications à son sujet dans l'aide en ligne ou sur msdn. Ce n'est pas ici notre sujet prioritaire.



Les éléments en gras et au pluriel sont des collections. On identifie ici les conteneurs de données : tables et vues, les procédures stockées, et la gestion des de la sécurité avec les groupes et utilisateurs.

Nous n'irons pas plus loin sur ADOX pour nous attarder sur ADO. Ce modèle fournit un service aux applications clientes (VB, Office...), mais il ne permet pas un accès direct à une base de données. Il faut pour cela s'appuyer sur des APIs de plus bas niveau telles qu'OLE DB.

5.4.3 *Les références*

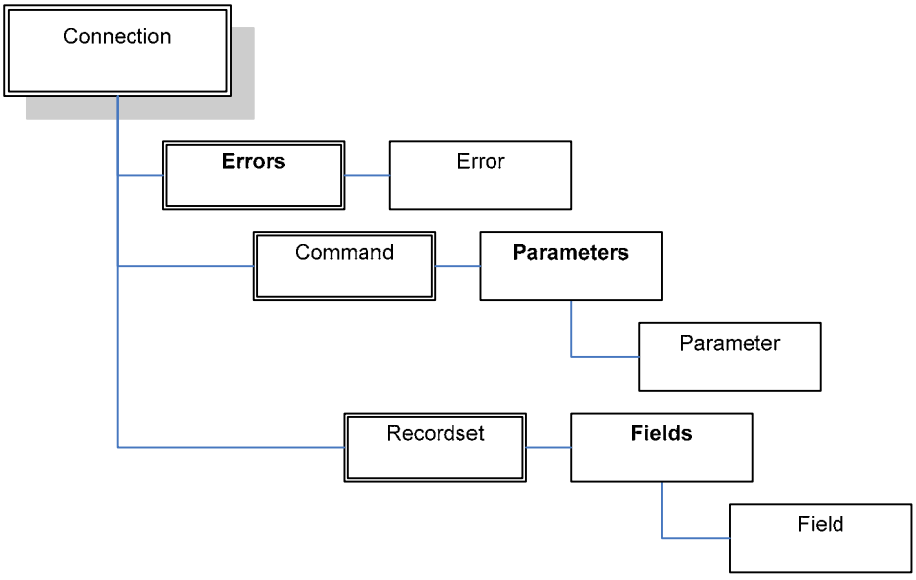
Pour accéder à la librairie ADO, il faut la mettre en référence dans un projet VB pour exploiter les objets qu'elle contient. Il s'agit de « ActiveX Data Objects. »

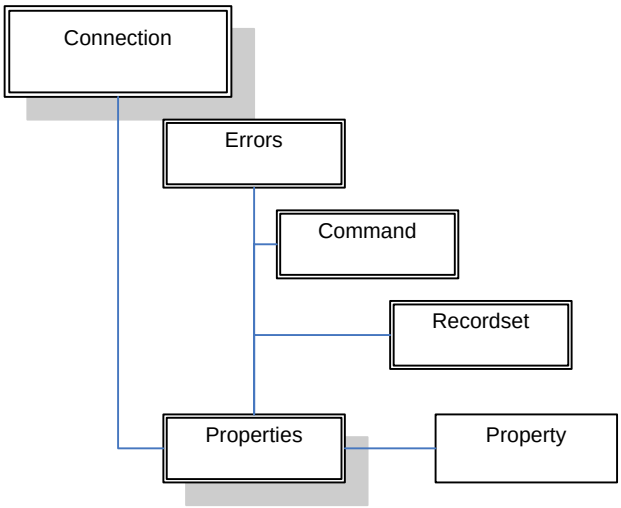
Le menu « Projet / Référence » affiche les bibliothèques installées sur le poste. Il suffit alors de cocher la case correspondante. L'explorateur d'objet (F2) affiche désormais ADO.

5.5 Les objets de ADO

5.5.1 *Hiérarchie des classes*

Voici le contenu de la librairie ADO. Comme vous pouvez le constater, les classes sont hiérarchisées.





↳ **Connection** établit une relation avec un fournisseur OLE DB pour accéder à une base de données.

↳ **Command** permet d'envoyer des commandes à la base de données sous forme de requêtes SQL ou d'appels de procédures stockées.

↳ **Recordset** permet de manipuler des données au sein d'un jeu d'enregistrements renvoyé par une requête « SELECT » ou une procédure stockée.

↳ **Parameters** est utilisé avec l'objet command lorsque la requête qu'il contient attend des paramètres.

↳ **Errors** et les objets « Error » qu'elle contient sont accessible exclusivement par l'objet « connection. » Il diffère de l'objet « Err » de VB car il ne remonte que les erreurs renvoyées par le fournisseur. Cela permet de récupérer des messages plus précis.

↳ **Fields** et les objets « Field » contenu sont accessibles par le Recordset lorsqu'il contient des données. Les champs sont créés automatiquement lorsqu'un jeu d'enregistrements est constitué dans le Recordset.

↳ **Properties** et les objets « Property » fournissent des informations supplémentaires sur les caractéristiques des objets qu'elles décrivent.

5.5.2 Connexion à une base

Il faut recourir à un fournisseur OLE DB pour établir une connexion à la base. Selon la base à laquelle on doit se connecter, on choisit le fournisseur OLE DB en passant le paramètre correspondant.

Les différents fournisseurs OLE DB sont fournis par les éditeurs de SGBD.

```
Private Sub ExempleConnection()
Dim cnx As New ADODB.Connection
```

```
    'Définition de la chaine de connexion
    'Ici le mot de passe est vide
    cnx.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" _
            & "User ID = Admin;Password = ;" _
            & "Data Source=E:\AIGEM\CoursAutoForm\Essai\ExempleADO.mdb;" _
            & "Persist Security Info=False"

    cnx.Open 'Ouverture de la connxion

    If cnx.State = adStateOpen Then 'Si son état est "ouvert", on affiche un message
       MsgBox "Connexion ouverte"
    End If

    cnx.Close 'Fermeture de la connexion

End Sub
```

La propriété *ConnectionString* est une chaine contenant les paramètres de connexion. L'exemple précédent illustre une connexion à une base Access. Les paramètres passés varient selon la base cible qui est précisée dans le *Provider*.
La liste des paramètres de l'exemple n'est pas exhaustive.

L'objet *Connection* est essentiel car il va être le vecteur de manipulation des données par ADO. En effet, ADO passe par la connexion pour récupérer des enregistrements dans des *Recordset*.

5.5.3 Les jeux d'enregistrements (RecordSets)

Cette classe permet d'accéder à des enregistrements sous forme de matrice (liste). On peut ensuite les manipuler, se déplacer dans le jeu d'enregistrements, faire des recherches…

Reprenons la connexion ouverte dans l'exemple précédent :

```
Dim rst as New ADODB.RecordSet
```

L'instruction suivante ouvre un jeu d'enregistrements :

```
rst.Open "Table", cnx, adOpenStatic, adLockReadOnly
```
ou
```
rst.Open "SELECT * FROM MaTable", cnx, adOpenStatic, adLockReadOnly
```

Le premier paramètre est la *Source*. Il peut s'agir d'un nom de table, de vue ou d'un ordre SQL *Select*.

Le second est l'objet connexion. Elle doit être ouverte et permet au *Recordset* de savoir dans quelle base récupérer les données. Un projet peut avoir plusieurs objets *Connection* ouverts.

Le troisième définit le type de jeu d'enregistrements. Les quatre valeurs possibles sont décrites par des constantes :

- *adOpenDynamic* : accès en lecture / écriture, les mises à jour des autres utilisateurs sont répercutées dans le *Recordset*. Ce type est plus lent ;
- *adOpenForwardOnly* : ce type est rapide et en lecture seule. Il ne peut être parcouru qu'en avant, du premier au dernier enregistrement ;
- *adOpenKeyset* : similaire aux *Recordset*s dynamique, ils n'affichent pas les modifications effectuées par les autres utilisateurs sur la source de données ;
- *adOpenStatic* : en lecture seule, ce type permet des déplacements en avant et en arrière.

Le Recordset est ensuite fermé et déréférencé.

```
rst.Close
Set rst = Nothing
```

5.5.4 Utilisation des RecordSets

Lorsqu'on ouvre un *Recordset* à l'aide d'une instruction SQL par exemple, nous ne savons pas si des enregistrements sont retournés. Le *recordset* peut être vide.
Il existe deux propriétés qui précisent si nous sommes en début de *Recordset* (BOF) ou en fin de *Recordset* (EOF). Si un *Recordset* ne contient aucun enregistrement, ses propriétés EOF et BOF sont à vrai simultanément.

↳ Déplacement :
Ces méthodes des *Recordsets* permettent de déplacer le curseur, c'est-à-dire changer l'enregistrement courant.

Les méthodes *Move*

```
Rst.MoveNext        enregistrement suivant
Rst.MovePrevious    enregistrement Précédent
Rst.MoveFirst       premier enregistrement
Rst.MoveLast        dernier enregistrement
Rst.Move n          déplacement de n enreg. En avant si positif, en arrière si négatif.
```

Ex : parcours d'un *recordset*

```
rst.Open "SELECT * FROM MaTable", cnx, adOpenStatic

If Not rst.EOF Then                 'Test s'il y a des enreg. Evite l'erreur
        rst.MoveFirst               'provoquée par Movefirst si aucun enreg.
        Do Until rst.Eof
                …[instructions]
                rst.MoveNext        'Avance d'un enreg à chaque tour
        Loop
End If

Rst.Close           'Fermeture du Recordset

Set rst = Nothing   'Déréférencement de la variable objet
```

Attention : les méthodes de déplacement du curseur échouent si le *Recordset* ne contient pas d'enregistrement. Il faut donc vérifier qu'il y a au moins un enregistrement avant toute manipulation des données.

↳ Recherche d'enregistrement :

Deux méthodes de l'objet *Recordset* sont disponibles pour rechercher des enregistrements : *Find* et *Seek*. La principale différence est que la méthode *Find* utilise des combinaisons de critères qui s'appuient sur tous les champs du *Recordset*, alors que la méthode *Seek* s'appuie sur un *Index* qu'il est nécessaire de définir à l'avance.
Dans les deux cas, le premier enregistrement répondant aux critères devient l'enregistrement actif. Si aucun enregistrement ne satisfait aux critères, la propriété *EOF* passe à vrai et aucun enregistrement n'est activé.

La méthode *Find…*

```
Find (criteria, SkipRows, searchDirection, start)
```

- *Criteria* : Chaine de caractère contenant une instruction spécifiant le nom de colonne, l'opérateur de comparaison et la valeur à utiliser pour la recherche (similaire à la clause WHERE du SQL) ;
- *SkipRows* : Entier long facultatif dont la valeur par défaut est zéro, qui spécifie le décalage à partir de la ligne en cours ou le signet *start* pour commencer la recherche ;
- *searchDirection* : Valeur *SearchDirectionEnum* facultative qui spécifie si la recherche doit commencer dans la ligne en cours ou dans la ligne suivante disponible, dans le sens de la recherche. Sa valeur peut être *adSearchForward* ou *adSearchBackward*. La recherche s'arrête au début ou à la fin du jeu d'enregistrements, en fonction de la valeur de *searchDirection* ;
- *Start* : Signet facultatif de type *Variant* indiquant la position de début de la recherche.

L'opérateur de comparaison spécifié dans *criteria* peut être ">" (supérieur à), "<" (inférieur à), "=" (égal) ">=" (supérieur à ou égal), "<=" (inférieur à ou égal), "<>" (différent de) ou "like" (comme en SQL).
La valeur indiquée dans *criteria* peut être une chaîne, un nombre en virgule flottante ou une date. Les valeurs de chaîne sont délimitées par des apostrophes (par exemple, "state = 'WA'"). Les valeurs de date sont délimitées par des signes dièse "#" (par exemple, "start_date > #22/07/97#").

Si l'opérateur de comparaison est "like", la valeur de la chaîne peut contenir "*" (une ou plusieurs occurrences d'un caractère quelconque) ou "_" (une seule occurrence d'un caractère quelconque). (Par exemple, "state like M_*" génère le résultat Maine et Massachusetts).

Ex :

```
Dim strCritere As String
```

```
Dim rst As New ADODB.Recordset

rst.Open "T_Utilisateur", gcnx, adOpenDynamic, adLockOptimistic 'ouverture du recordset

strCritere = "Nom = ""Toto""" 'Construction du critère

rst.Find strCritere, , adSearchForward 'Recherche de l'enregistrement correspondant au critère

If Not rst.EOF Then 'Si trouvé, on met à jour le nom
   rst.Fields("Nom").Value = "Titi"
   rst.Update
Else
   MsgBox "Enregistrement introuvable."
End If

rst.Close 'Fermeture et libération
Set rst = Nothing
```

↳ Accès aux champs d'un Recordset :

Plusieurs syntaxes équivalentes permettent d'accéder aux champs d'un Recordset.

```
rst.Fields(0).value = "Toto" 'Accès par l'index du champ dans la collection fields.
rst.Fields("Nom").value = "Toto" 'Accès par le nom du champ dans la collection fields.
```

Comme Value est la propriété par défaut, on peut écrire :

```
rst.Fields(0) = "Toto"
rst.Fields("Nom") = "Toto"
```

Comme Fields est la collection par défaut, on peut écrire :

```
rst(0) = "Toto"
rst("Nom") = "Toto"
```

Autres notations :

```
rst!Nom = "Toto"
rst![Nom] = "Toto"
```

La lecture d'un champ se fait en plaçant la référence au champ à droite de l'affectation, puisqu'il s'agit d'une propriété d'objet.

↳ Ajout d'enregistrements :

L'ajout d'enregistrement est possible dans des *Recordsets* dont le type de curseur autorise l'écriture. Il s'agit de curseurs *Dynamic* et *Keyset*.
Il faut aussi que les enregistrements n'enfreignent pas les règles d'intégrités de la table ou requête sources du *Recordset*, comme des doublons sur une clé primaire ou des clés étrangères.

On utilise la méthode *AddNew* du *Recordset* pour définir un nouvel enregistrement. On peut alors renseigner les champs de l'enregistrement. Toutefois, l'ajout de ne sera effectif qu'après la validation effectuée par la méthode *Update*. La méthode *CancelUpdate* libère l'enregistrement et annule l'ajout ou les modifications.

Ex :

```
rst.Open "T_Utilisateur", gcnx, adOpenDynamic, adLockOptimistic 'ouverture du recordset
rst.AddNew
        Rst("NOM") = "Jourdain"
        Rst("PRENOM") = "Robert"
rst.Update
```

↳ Modification d'enregistrements :

La modification fonctionne avec les mêmes contraintes que précédemment. Il faut d'abord se déplacer sur l'enregistrement à modifier pour le rendre actif. On affecte les valeurs souhaitées aux champs du *Recordset*. La validation se fait avec les mêmes méthodes que pour l'ajout.

Ex :

```
rst.Open "T_Utilisateur", gcnx, adOpenDynamic, adLockOptimistic 'ouverture du recordset

        Rst("NOM") = "Jourdain"
        Rst("PRENOM") = "Robert"
rst.Update
```

↳ Suppression d'enregistrements :

La suppression agit sur l'enregistrement courant. On invoque la méthode *Delete* du *Recordset*. Le curseur n'est pas repositionné automatiquement, il faut donc le faire avec une des méthodes de déplacement. Il faut veiller à gérer le cas de la suppression du dernier enregistrement car les méthodes de déplacement échouent sur un *RecordSet* vide.

Ex pour effacer tout un Recordset :

```
Do Until rst.EOF
        rst.Delete
        rst.MoveNext
Loop
```

↳ Autres propriétés et méthodes :

Il existe de nombreuses propriétés et méthodes dans la classe *Recordset*. Une méthode utile avec les *recordsets* d'un type différent de Table est *Requery*. Ces *recordsets* correspondant à une requête de sélection, il est possible de les rafraîchir en ré interrogeant la source dans le cas ou celle-ci serait modifiée par ailleurs.

La propriété *RecordCount* est aussi très utile. Elle retourne le nombre d'enregistrements contenus dans un *recordset*. Attention, elle retourne les enregistrements accédés uniquement, or ceci se fait au travers d'un buffer de lecture. Il faut donc accéder au dernier enregistrement à l'aide de la méthode *MoveLast* pour que *RecordCount* contienne le nombre total d'enregistrements.
RecordCount est réinitialisée après *Requery*.

5.6 Développement visuel

Nous n'étudierons pas en détail le développement visuel car il n'est pas utilisé pour les applications professionnelles. Bien qu'il soit très rapide à réaliser, il peut manquer de souplesse pour le développement, et les performances ne sont pas optimales du fait même de l'architecture qui nécessite une connexion permanente sur le jeu d'enregistrements.

Le principe de base est, lors de la conception et graphiquement, de configurer les objets permettant d'établir une connexion à une base de données, puis à un jeu d'enregistrement et à un champ.

Lors de l'exécution, le contrôle final présenté sur un formulaire pointe sur une table ou une requête et l'utilisateur peut consulter ou mettre à jour des données selon le paramétrage positionné par le développeur. Tant que l'écran est ouvert par un utilisateur, le jeu d'enregistrements sources est ouvert. Cela peut engendrer des difficultés de travail coopératif (accès concurrents) ou de temps de réponse.

Il est important de souligner que les objets accédés par les contrôles sont les mêmes que ceux décrits précédemment. Il s'agit juste de les mettre à disposition du développeur à l'aide d'éléments graphiques.

5.6.1 Les propriétés du contrôle ADOData

Ce contrôle permet d'établir un lien vers la base de données, jusqu'à un jeu d'enregistrements. Il permet aussi la navigation dans le jeu d'enregistrement et la configuration des accès aux données.

Le premier service permet de se connecter à une base. On peut utiliser différents fournisseurs selon la base souhaitée (SQL Server, Oracle, ODBC, Access…)

On utilise la propriété *ConnectionString* pour atteindre la base de données ; la même que l'objet *Connection* évoqué au paragraphe précédent.
On positionne ensuite les propriétés *CommandType* puis *RecordSource* qui décrit un jeu d'enregistrement (table, vue, procédure stockée.)

5.6.2 Les contrôles liés

Il s'agit ici des contrôles que l'on peut lier à un contrôle *ADOData* pour modifier les données sources. Ces contrôles sont dits dépendants lorsqu'ils sont connectés à un contrôle *ADOData*. En voici quelques-uns, cette liste n'est pas exhaustive :

Label, TextBox, CheckBox, PictureBox, Image, ComboBox, ListBox, OLE, DBCombo, DBList, DBGrid, MSFlexGrid, MaskEditBox, …

On utilise les deux propriétés suivantes pour lier un contrôle à un champ via un ADOData :

DataSource : nom du contrôle *ADOData*
DataField : nom du champ du contrôle *ADOData* ou du *Recordset* lié dans le cas d'un développement non visuel.

Les contraintes de mise à jours des données sources sont les mêmes que pour le développement objet (requêtes multi-tables avec jointures…).

5.6.3 Les contrôles liés complexes

Certains contrôles de manipulation de données sont plus sophistiqués. Il s'agit souvent de variantes de contrôles classiques pourvus de propriétés ou méthodes supplémentaires.
On trouve ainsi une liste déroulante (DBCombo), une zone de liste (DBList), une grille de données (DBGrid ou MsFlexGrid)… De nombreux éditeurs développent des contrôles ActiveX d'accès aux données plus complets que ceux livrés par Microsoft en standard.

Leur utilisation est simple pour des fonctionnalités de base, mais peut se compliquer rapidement. Reportez-vous à l'aide en ligne de VB pour connaître l'utilisation de ces contrôles. Les contrôles, du moins ceux du commerce, sont livrés avec une documentation décrivant leurs propriétés, méthodes et événements.

L'essentiel

Nous avons traité ici un élément important du développement avec Visual Basic : la manipulation de données.

La méthode utilisée quasiment toujours consiste à manipuler les objets d'accès aux données par programme. Le développement visuel est très rare, jusqu'à être absent des applications professionnelles. Cela étant, le développement visuel fait « simplement » appel à une couche graphique supplémentaire pour agir sur les objets d'accès aux données.