Visual Basic et les Bases de Données
(Résumé)

Le concept de « Base de Données » est largement utilisé dans tous les domaines qui impliquent l'informatique. Visual Basic est, par excellence, un langage qui facilite le travail avec ce concept. Avant de commencer à présenter les liaisons entre VB et les bases de données, faisons une brève introduction au concept de base de données.

Une base de données est une structure spécialement conçue pour la gestion de grandes quantités de données. Par gestion on comprend l'ajout, la suppression, la modification et l'utilisation (calculs, manipulations, sélections, affichage, etc..) de données de différents types. D'une base à l'autre, les types de données peuvent être différents. Par exemple dans une base de type Access il existe le type booléen (par exemple Vrai/Faux ou Oui/Non, etc.) tandis que dans une base de type MSSQL ce type n'existe pas et, donc, il faudrait le remplacer par un autre – char*1 ("V"/"F") ou même entier (0/-1). La façon d'enregistrer les données ainsi que la façon de lire les données peut différer d'un type de base à l'autre. À cause des différences entre les bases de données, la manière avec laquelle on travaillera sera aussi différente.

Dans ce texte, nous allons analyser les liaisons entre VB et les bases de données relationnelles. Parmi les bases de données relationnelles nous prendrons l'exemple de Access.

La base «BaseTP4.MDB»

Cette base de données est constituée de 3 tables, trois ensembles de données regroupées, l'une contenant les données sur les étudiants inscrits, une autre sur les départements d'attache de ces étudiants et la dernière sur les groupes où sont classés les étudiants.

La table **IFT1175** :

Nom du champ	Type de données
NOM	Texte (maximum 33 caractères)
CODEPERM	Texte (maximum 12 caractères)
GROUPE	Octet
Num DEPT	Entier long (= entier en VB.NET)
REMARQUE	Texte (maximum 20 caractères)
INTRA	3 données numériques (réelles doubles)
FINAL	
TP	

On notera que la fiche de l'étudiant contient le «numéro» du département plutôt que le nom du département et le numéro de groupe plutôt que sa description. Notez aussi que le nom du champ «Num Dept» contient un espace, ce qui est permis en Access.

La table **Fac-Dept** :

Nom du champ	Type de données
NumDEPT	NuméroAuto. Correspond au champ «Num Dept» de la table «IFT1175»
DEPT	Texte (max. 25 caractères) indiquant le nom du département.

La table **Groupes** :

Nom du champ	Type de données
NumGroupe	Entier long. Correspond au champ «GROUPE» de la table «IFT1175»
Description	Texte (max. 20 caractères) décrivant le groupe : ex. «Mardi AM».

On pourra donc aller chercher des données soit dans une table (ex. «Liste des nom des étudiants avec leur note de l'intra») ou dans plusieurs tables (ex. «Liste des étudiants dont le nom du département est "Chimie"» ou bien «Liste des étudiants (Nom et Code permanent) avec le nom de leur département et l'horaire de leur cours».

Communication avec les bases de données

Afin d'établir la communication entre les applications et les bases de données, les langages de programmation mettent à la disposition du programmeur, différents outils. VB est un langage qui offre plusieurs outils simples mais efficaces dans le but de faciliter la programmation et l'utilisation de telles liaisons.

Chaque type de base de données est associé à un «**fournisseur**», c'est-à-dire un logiciel qui répond aux requêtes qui lui sont soumises en traitant la structure **et** le contenu de la base. Il existe différents «**fournisseurs**» d'accès aux bases de données. La liaison avec une base de données de type Access se fait à l'aide du ***moteur de recherche*** **Microsoft Jet OLE DB Provider** qui gère les bases Access. Ceci n'est pas laissé au choix de l'usager car le type de connexion et le langage de requêtes dépendent du type de base qu'on utilise. Ainsi, les fichiers Access 2003 (version 11) utilisent la version «**Jet 4.0**» alors que les bases de Access 97 (version 8) utilisent la version **3.51** du «**Jet**».

Pour établir des connections avec des bases de données, VB nous offre une large variété de contrôles graphiques ainsi que des classes d'objets non graphiques. Certains de ces contrôles ont été créés spécifiquement pour gérer ces liaisons. D'autres sont des contrôles que vous connaissez déjà et qui contiennent simplement des propriétés qui facilitent la liaison avec les données qui proviennent d'une base de données.

DAO, ADO et ADO.NET

Les versions 5 et antérieures de VB utilisaient des outils de connexion rassemblés sous le vocable de **DAO** (**Data Access Objects**) alors que la version 6 offrait des outils plus performants regroupés utilisant la technologie «OleDb» et regroupés sous le vocable **ADO** (**ActiveX Data Objects**). VB.NET ne supporte plus le DAO mais continue de supporter le ADO pour fins de continuité. Mais l'accent pour les nouvelles applications est mis sur une

version mise à jour (améliorée) de ADO : les outils **ADO.NET** qui peuvent gérer les types de bases les plus variées, incluant même des fichiers texte, Excel, Serveurs SQL, pages Web, intranets, banques d'images, etc.

Microsoft recommande donc d'utiliser le modèle ADO.NET puisque les outils ADO seront abandonnés éventuellement. Ceci dit, toute la structure de ces objets est bicéphale : la partie «OleDb» pour la plupart des fournisseurs de données actuels et la partie SQLClient spécialisée pour la gestion des bases fournies par la version 7 (et ultérieure) du serveur SQL de Microsoft.

Tous les outils de traitement des données sont contenus dans l'espace de nom «System.Data» qui apparaît par défaut dans les références de départ. Pour accéder aux outils «OleDb», il suffit de mentionner au début d'un module :

```
Imports System.Data.OleDb
```

Classes OleDb

L'ensemble de la technologie OLE DB, intégrée au FrameWork.NET, permet au programmeur de créer des commandes qui sont indépendantes des bases de données utilisées en passant par les différents objets intermédiaires qui servent alors d'interface. On change de base ? On ajuste les interfaces et on n'a pas à modifier le programme.

Classe OleDbConnection

Un objet de cette classe possède une propriété fondamentale de type «String», la «ConnectionString» qui décrit à la fois le type de liaison (le ***moteur de recherche*** et le fichier ou base à ouvrir). Supposons que la variable «String» contient un texte adéquat, alors on peut écrire

```
Dim UneConn As OleDbConnection
UneConn = New OleDbConnection
UneConn.ConnectionString = UneChaine
UneConn.Open()    ' Méthode qui «ouvre» la connexion.
```

ou, plus brièvement, en utilisant le constructeur «New»

```
Dim UneConn As OleDbConnection = New OleDbConnection(UneChaine)
UneConn.Open()    ' Méthode qui «ouvre» la connexion.
```

qui ouvrira alors la connexion. En cas d'erreur dans la chaîne de correction, il y aura interruption du programme avec un message de VB.

À partir de ce moment, on peut utiliser deux ou trois classes d'objets pour, ou bien lire les données de la base, ou bien les modifier et les mettre à jour.

Après la lecture des données, on peut bien sûr fermer la connexion avec

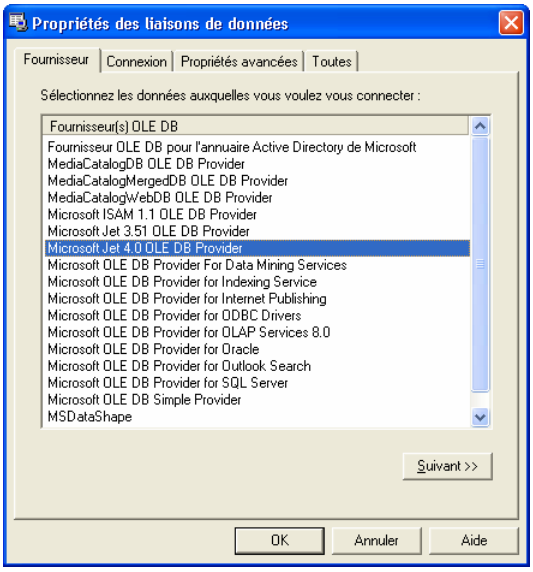
```
UneConn.Close()
```

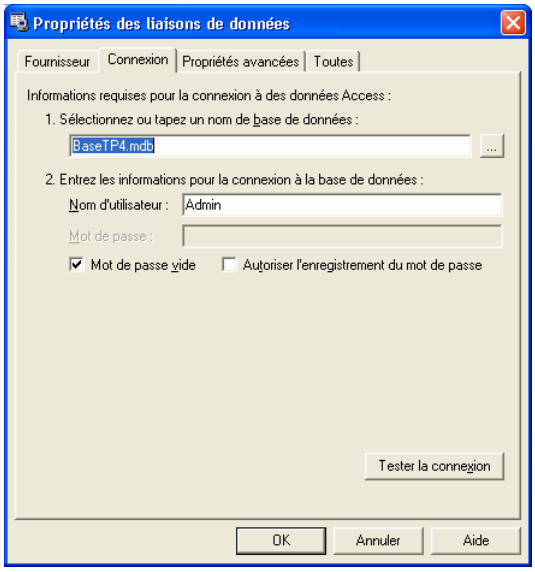
Propriété ConnectionString

Si on ne connaît pas le libellé de la chaîne de connexion adéquate, on peut utiliser un outil développé en VB6 pour la définir facilement : le fichier «XXX.udl». Un fichier de type «*Universal Data Link*» contient une chaîne de connexion qui peut être modifiée par un éditeur «Assistant des liaisons de données.

On peut créer une liaison en suivant ces étapes :

1. On crée (bouton droit de la souris) un «Nouveau» / «Document texte» qui portera le nom «Nouveau document texte.txt».
2. On renomme le fichier, «*UnNom*.udl» pour en faire un fichier de liaison vide.
3. On double-clique sur le fichier pour lancer l'éditeur de liaison de données.
4. Dans la boîte de dialogue, deux onglets définissent le moteur de recherche (onglet «Fournisseur» et le fichier lui-même (onglet Connection».





5. Selon le type de base, les autres onglets serviront à définir l'environnement de l'accès, comme le mot de passe, les permissions partagées, etc. Dans le cas d'une simple base Access, les deux premiers onglets peuvent suffire.
6. Dans l'onglet «Fournisseur», on sélectionne «**Microsoft Jet xx OLE DB Provider**».
7. Dans l'onglet «Connexion», on peut taper le nom de la base ou bien naviguer sur l'ordinateur (bouton [...]) pour localiser la base. Si on désire que le fichier soit en adressage relatif, il faut supprimer le chemin pour ne laisser que le nom.
8. Après avoir fermé l'application, il suffit d'ouvrir le fichier avec **WordPad**, **NotePad** ou tout autre éditeur de texte et vous y trouverez la chaîne de connexion qu'il suffira de «Copier/Coller» dans votre programme VB :

```
[OleDb]
; Everything after this line is an OLE DB initstring
Provider=Microsoft.Jet.OleDb.4.0;Data Source=BaseTP4.mdb;Persist Security Info=False
```

Vous noterez que le nom du fichier peut varier d'une application à l'autre. C'est pourquoi on peut séparer la chaîne en trois sections : le nom de la base, la partie qui le précède et la partie qui le suit. On peut alors reconstituer la chaîne en concaténant les trois parties avant de l'utiliser.

Classe OleDbCommand

Une base de données, par définition, contient de nombreuses données de sorte qu'une «interrogation» consiste à soumettre une «requête» spécifiant les informations désirées. Le langage «standard» d'interrogation des bases de données remonte aux années 70 et continue de dominer le «marché» des SGBD (système de gestion de bases de données). Il s'agit du SQL (*Structured Query Language* ou Langage de requêtes structuré).

Une requête SQL est une chaîne de caractères (ex. «SELECT * FROM XXX» où XXX est le nom d'une table de la base voir plus haut).

Si on présume que la chaîne «UneReqSQL» contient une requête valide en fonction de la base de données visée par la commande, un objet de la classe «OleDbCommand» peut associer cette requête SQL à un objet «OleDbConnection comme suit :

```
Dim UneCommande As OleDbCommand
UneCommande = New OleDbCommand
UneCommande.Connection = UneConn
UneCommande.CommandText = UneReqSQL
```

ou, plus brièvement, en utilisant le constructeur «New»

```
Dim UneCommande As New OleDbCommand(UneReqSQL, UneConn)
```

Une fois que la commande est définie (requête plus connexion), on doit appeler une méthode (parmi plusieurs) permettant de «lancer» (exécuter) cette requête en vue de recevoir les résultats demandés.

Classe OleDbDataReader

L'objet de classe «OleDbDataReader» est une interface vers les données permettant d'accéder séquentiellement aux données obtenues à partir de l'objet précédent, de classe OleDbCommand. Il s'utilise avec une requête dont l'effet est de ramener un jeu d'enregistrements, c'est-à-dire une séquence de lignes d'une table dont les colonnes sont les champs de la base de données. On l'obtient comme suit :

```
Dim UnDataReader As OleDbDataReader
UnDataReader = UneCommande.ExecuteReader()
```

L'objet «DataReader» fonctionne tout à fait comme un objet de classe «StreamReader» : il possède une méthode booléenne «Read()» et chaque appel à cette méthode place le prochain enregistrement dans une variable cachée (de type *Collection*) à laquelle nous donne accès la propriété publique «Item(*index*)».

Par exemple, supposons que la requête
«SELECT CodePerm, Nom, Intra, Final FROM IFT1175 WHERE [Num DEPT] = 12»
placée dans l'objet «UneCommande» ci haut ait pour effet de ramener le bloc d'enregistrement suivant de la base de données reliée, soit trois enregistrements des quatre champs «CodePerm», «Nom», «Intra» et «Final».

GROG25127002	Grondin, Gontran	89	78
OUEP25627020	Ouellette, Paulette	86	91
LAUR25017418	Laurent, Rolland	61	58

Alors la séquence d'énoncés suivante ouvrira une connexion à la base de données puis, dans une boucle de trois itérations, affichera le nom et la moyenne des deux examens dans une boîte de message. Notez que la propriété «Item» étant la propriété de défaut, l'expression «`UnDataReader(3)` équivaut à `UnDataReader.Item(3)`

```
Dim UneChaineSQL As String = "SELECT CodePerm, Nom, Intra, Final " _
     & "FROM IFT1175 WHERE  [Num DEPT] = 12"
Dim UneChaineConn As String = _
     "Provider=Microsoft.Jet.OleDb.4.0;"  _
     & "Data Source=BaseTP4.mdb;" _
     & "Persist Security Info=False"
Dim UneConn As OleDbConnection = New OleDbConnection(UneChaine)
UneConn.Open()    ' Méthode qui «ouvre» la connexion.
Dim UneCommande As New OleDbCommand(UneChaineSQL, UneChaineConn)
Dim UnDataReader As OleDbDataReader
UnDataReader = UneCommande.ExecuteReader()
Dim UnMess As String, UneMoy As Single
Do While UnDataReader.Read()  '  sous-entendu = True
   UnMess = UnDataReader.Item("Nom") & " a une moyenne de : "
   UneMoy = (UnDataReader(2) + UnDataReader.Item(3)) / 2
      ' équivaut à UnDataReader("Intra") et à UnDataReader("Final")
   UnMess = UnMess & UneMoy.ToString
   MsgBox(UnMess)
Loop
UnDataReader.Close()
UneConn.Close()
```

Sommaire des propriétés et méthodes des classes précédentes.

Objet	Type	Membre	Description
OleDbConnection	Propriété lect-écr	ConnectionString	Chaîne de texte définissant la source d'information : fournisseur et base.
	Méthode Sub	Open()	Ouvre la connexion.
	Méthode Sub	Close()	Ferme la connexion.
OleDbCommand	Propriété lect-écr	CommandText	Chaîne de texte contenant la requête SQL.
	Propriété lect-écr	Connection	Objet de type OleDbConnection.
	Méthode Objet	ExecuteDataReader()	Exécute la commande et retourne un objet de classe OleDbDataReader
	Méthode Objet	ExecuteScalar	Exécute la commande et retourne la première valeur obtenue de la base de données. S'utilise avec une requête qui va chercher une seule valeur comme une somme ou un décompte dans les données.
	Méthode Integer	ExecuteNonQuery	Exécute la commande et retourne le nombre de lignes affectées. S'utilise avec une requête qui modifie, ajoute ou supprime des données.
OleDbDataReader	Propriété lect.	FieldCount	Indique (booléen) le nombre de champs (colonnes) disponibles via la propriété *Item*.
	Propriété lect.	HasRows	Indique (booléen) si le nombre d'enregistrements est supérieur à zéro.
	Propriété lect.	Item(*Index*)	Obtient la valeur de la colonne *Index*.
	Méthode Boolean	Read()	Tente de «lire» le prochain enregistrement de la base et retourne Vrai ou Faux selon que la lecture a eu lieu ou non.
		Close()	Ferme l'objet mais non la connexion.

Visual Basic et les Bases de Données
(Mises à jour)

L'utilisation d'une base de données sous-entends généralement la gestion d'un grand nombre d'informations regroupées d'une façon plus ou moins complexes selon les applications développées. Le rôle de ADO.NET est de fournir les outils permettant d'accéder à ces données et, éventuellement les mettre à jour, soit en les modifiant, soit en les créant, soit en les supprimant selon les besoins.

Nous avons vu précédemment que pour accéder aux données, les objets de classes **OleDBConnection**, **OleDBCommand**, et **OleDBReader** permettaient de «lire» la base de données tout comme on lit un fichier séquentiel, soit un enregistrement (une ligne(à la fois, avec la possibilité, avant ou après chaque «lecture» de vérifier si la liste est épuisée.

Mais tout comme le cas des fichiers à accès séquentiels, le **DataReader** ne permet que la lecture des données et non leur mise à jour. Par contre, un «habitué» de la programmation des bases de données (donc un «Expert» en SQL peut, à partir de l'objet de classe **OleDBCommand** envoyer à la base de données toutes les commandes nécessaires pour **ajouter**, **supprimer** ou **modifier** des enregistrements. En effet, les deux méthodes **ExecuteScalar** et **ExecuteNonQuery** de cette classe permet d'envoyer tout le jeux des commandes SQL possibles. Mais attention, il faut alors programmer chacune des étapes du programme.

Le principe de la mise à jour

Dans une application Windows, l'un des éléments les plus importants est l'utilisation des contrôles graphiques (zones de textes, ComboBox, étiquettes, etc.) comme interface entre le programme et l'utilisateur : celui-ci interagit avec ces contrôles et le programme gère les informations en conséquence. Dans le cas de la mise à jour des données, il faut donc établir le lien entre l'information qui se trouve dans la base de données et l'information qui se trouve dans les contrôles de la feuille.

VB.NET, utilise donc les outils de ADO.NET pour placer dans les contrôles les informations de la base et à l'inverse, ramener dans la base les valeurs placées par l'utilisateur dans les contrôles. Or les informations placées dans les sources de données (base Oracle, SQLServer, fichiers Excel, Quattro Pro, etc.) ont des formats différents et donc demanderaient des codes de programme différents.

ADO.NET fonctionne donc comme ceci : une fois établie la connexion avec la source des données (quelle qu'elle soit), une copie de ces informations sera créée dans une zone de mémoire de l'application, c'est-à-dire un «DataSet». Cette copie ayant un format unique, le programme s'écrira donc de la même façon peu importe le format des données dans la source. Entre la source et le «Dataset» ADO.NET verra à transférer les informations correctement et le programmeur n'aura qu'à gérer le lien entre le «Dataset» et ses contrôles d'affichage sur la feuille.

Il n'existe pas (comme en ADO de VB6) de super outil qui fasse le lien directement entre la base et les contrôles (probablement pour une prochaine version). Il faut donc utiliser et relier plusieurs objets pour définir les différents segments de ces liens. Voici donc les objets qui pourront être utilisés pour établir ces liens à «plusieurs étages».

Schéma général

On peut schématiser les éléments qui vont suivre dans une chaîne progressive entre les données de la base et les contrôles de la feuille :

Connexion ↔ DataAdapter ↔ DataSet ↔ DataTable ↔ DataView ↔ DataBinding ↔ Contrôle

Ou bien dans certains cas :

Connexion ↔ DataAdapter ↔ DataSet ↔ DataTable ↔ Contrôle

Classe OleDbConnection

On utilisera ici aussi l'outil de connexion déjà utilisé à l'étape précédente. Se rappeler simplement les deux éléments essentielles de la propriété principale (**ConnectionString**) de cet objet : le fournisseur de service et le nom de la base utilisée.

Dans le cas des mises à jour, ADO.NET prévoit que la connexion est établie, que les données sont transférées en mémoire et que la connexion est immédiatement fermée. Ainsi, on évite de surcharger le réseau dans le cas d'une base centralisée (ce qui est le plus souvent le cas dans les applications commerciales des bases de données). Une fois les mises à jour effectuées localement en mémoire, la connexion est ré ouverte, les données modifiées retransmises dans la base et la connexion refermée.

Classe OleDbDataAdapter

L'objet de cette classe (un *DataAdapter*) constitue le coeur de l'interface (le pont) entre la base de données et le programme. C'est celui qui transfère les information entre la base et l'application en convertissant au besoin les données de façon à ce qu'elles puissent être traitées localement **comme s'il s'agissait d'une base de données relationnelle**, quel que soit leur format dans la source. Les deux propriétés qui permettent d'aller chercher les données sont les mêmes que pour le «OleDbCommand», soit la connexion (OleDBConnection) et la chaîne SQL sous forme de variable ou constante, ou sous forme d'une commande **OleDbCommand**. :

```
Dim UnAdapt As New OleDbDataAdapter("unechaineSQL", UneConn)
Dim UnAdapt As New OleDbDataAdapter(UneCommande, UneConn)
```

La chaîne SQL utilisée sera placée dans la propriété «SelectCommand» et permettra de transférer les informations de la base de données vers la zone de mémoire où résidera la copie. La méthode «**Fill**» pourra effectuer ce travail (voir plus loin). Mais pour mettre à jour la base, il faudra en plus transférer les données dans l'autre sens (méthode «Update») et pour cela, il faudra définir les commandes SQL de suppression, d'ajout et de modification des enregistrements. On peut définir ces commande (si on les connait) dans les propriétés suivantes :

```
UnAdapt.DeleteCommand = "UneChaineSQL2"
UnAdapt.InsertCommand = "UneChaineSQL3"
UnAdapt.UpdateCommand = "UneChaineSQL4"
```

Une fois que les 4 chaînes SQL sont définies, les méthodes «Fill» et «Update» du DataAdapter transféreront les informations dans un sens et dans l'autre.

Classe DataSet

L'objet «Dataset» est un espace de mémoire qui contient une structure de base de données relationnelle, c'est-à-dire qu'il contient des tables de données et des «liaisons» entre ces tables. TOUTES les données se trouvent dans la ou les tables du DataSet et TOUT le travail de mise à jour s'y fera. Un DataSet simple ne contient qu'une table. Un DataSet complexe contient plusieurs tables. De fait, le DataSet contient une collection de tables nommée «Tables». Ainsi, l'accès à une table peut se faire par la propriété Tables :

```
Dim UnDataSet As DataSet
...  ' le contenu d'une base y est transféré
```

Un DataSet n'a pas de méthode pour «aller chercher» des données. Il «attend» qu'on (le DataAdapter) les lui donne «UnAdapt.Fill()». Une fois une ou des tables ajoutées à sa collection, on peut y accéder par le rang (base 0) ou par le nom. qu'on a donné à la table. Comme la propriété «Item» est la propriété de défaut de Tables, les énoncés suivants seront équivalents si le nom de la première table est «MaListe» :

```
Dim UneTable As DataTable = UnDataSet.Tables.Item(0)
Dim UneTable As DataTable = UnDataSet.Tables.Item("MaListe")
Dim UneTable As DataTable = UnDataSet.Tables(0)
Dim UneTable As DataTable = UnDataSet.Tables("MaListe")
```

Bref, dans la séquence suivante, on établit une connexion, on crée un DataAdapter, on crée une variable (espace) DataSet puis on traite les données. Ensuite, on met à jour la base de données en transférant les enregistrements modifiée.

```
Dim UneConn As OleDbConnection = New OleDbConnection(UneChaineConn)
UneConn.Open()
Dim UnAdapt As New OleDbDataAdapter("unechaineSQL", UneConn)
Dim UnDataSet As New DataSet("UnnomOK") '(Simple nom arbitraire)
UnAdapt.Fill(UnDataSet, "PremTable") ' Une table est ajoutée
UneConn.Close()
... traitement des données ...
UneConn.Open()
Dim NbEnrMod As Integer  ' variable entière
NbEnrMod = UnAdapt.Update(UnDataSet)   ' NbEnrMod indique le nombre
           ' de modifications
UneConn.Close()
```

Sauf que dans le cas précédent, pour que la méthode «Update» réussisse, il faut que les chaînes SQL appropriées aient été définies. La classe «**OleDbCommandBuilder**» permet de créer ces chaînes automatiquement. On ajoute donc avant d'utiliser «Updata» l'énoncé :

```
Dim UnBuid As New OleDbCommandBuilder(UnAdapt)
```

et le tour est joué.

Classe DataTable

Les objets «DataTable» sont des jeux d'enregistrements qui sont obtenus par une requête de sélection (extraction) SQL lors de l'exécution de la méthode «Fill». En général, si on désire obtenir plusieurs tables, on définira plusieurs objets DataAdapter qui contiendront chacun la chaîne SQL appropriée. Ainsi, sur la feuille «frmNotes» qui traitera des informations provenant des trois tables de la base de données, la chaîne SQL ira chercher les données non pas dans les tables de Access mais dans une requête déjà définie «Liste globale» et le résultat de la méthode «Fill» sera une seule table.

Une fois la table remplie, on peut accéder aux données au moyen de ses propriétés d'accès. Le jeu d'enregistrement est traité comme un tableau à deux dimensions formé soit d'un ensemble de lignes (propriété «Rows) de type collection contenant des champs, soit un ensemble de colonnes (propriété «Columns») de type collection contenant des enregistrements. Ainsi les deux énoncés suivants pourraient être équivalents si les noms de champs sont bien définis et désigneraient le «nom» contenu dans le premier enregistrement :

```
MaTable.Rows(0).Item("Nom") = "Grondin, Gontran"
MaTable.Columns("Nom").Item(0) = "Grondin, Gontran"
```

La programmation de chacune des «cellules» de la table pouvant être fastidieuse, deux autres objets permettent d'en faciliter la tâche.

Classe DataView

Les objets «DataView» sont aussi des jeux d'enregistrements. Mais une «vue» (comme une requête en Access) permet de réorganiser les données de la table par exemple en les triant, ou bien en filtrant des données. Exemple, ne prendre de la table que les enregistrements où le champ "Intra" est plus grand que le champ "Final" et «sortir» la liste en ordre alphabétique.

```
Dim UneVue As New DataView(MaTable)
UneVue.RowFilter = "Intra < Final"  ' Les champs doivent exister
UneVue.Sort = "Nom"
```

L'utilisation de UneVue se fera de la même manière que MaTable sauf que les enregistrements seront présentés différemmant.

Classe de liaison DataBindings

Jusqu'ici, les données obtenues sont confinées (sous différentes formes) en mémoire et ne sont aucunement visibles de l'utilisateur. Un objet «BindingManagerBase» peut synchroniser les données et les contrôles de façon à ce que, en indiquant une «position» dans le DataView, les informations des contrôles seront ajustés à ceux du DataView. Ainsi, il suffira de déplacer la valeur de position («avancer» ou «reculer» et les contrôles associés seront aussi ajustés, c'est-à-dire les contrôles visibles sur la feuille susceptibles de contenir des données : TextBox, Label, CheckBos, ComboBox, Scrollbars, etc.

```
Dim UnBind As New BindingManagerBase
UnBind = Me.BindingContext(UneVue, "")
```

créera un gestionnaire de liaisons associé à la propriété «`BindingContext`» **de la feuille** et qui jouera le chef d'orchestre entre les données de «UneVue et les contrôles de la feuille.

Le principe est d'établir un lien entre une colonne (un champ) de la vue et UNE propriété du contrôle. La plupart du temps, c'est la propriété «Text» qui est associée au champ en question mais ce pourrait être image, une couleur, une police de caractères, etc. Comme la feuille, c'est «ME», on construira donc la liste des liaisons en utilisant la propriété «**DataBindings**» de chacun des contrôles à associer :

```
txtCodeP.DataBindings.Add(New Binding("Text", MaDataVue, "CODEPERM"))
cboDept.DataBindings.Add(New Binding("SelectedIndex", MaDataVue, "Num DEPT"))
```

Ici, le «BindingManagerBase» placera la valeur courante du champ "CODEPERM" de la DataView dans la propriété «Text» du contrôle «txtCodeP» et vice-versa lorsque cette valeur aura été modifiée. Pour changer la position de l'enregistrement courant de la DataView, on utilise simplement la propriété «Position» du «BindingManagerBase».

Donc tout changement de cette propriété synchronisera les contrôles et l'enregistrement correspondant de la Vue.

Visual Basic et les Bases de Données
(Sans programmation)

En tradition avec les versions précédentes de Visual Basic, VB.NET offre des «outils automatiques» et des assistants pour les ajuster, le but étant de définir une connexion à la base sans avoir à la programmer. Pour ce faire, VB met à la disposition du programmeur une section de la boîte d'outils intitulée «Données» (*Data*) contenant la plupart des classes déjà vues sous forme de contrôles.

Plutôt que de créer la liste des objets à l'exécution, on peut définir chaque étape des la liaison avec les contrôles au moment du «design». L'«assistant» présente des boîtes de dialogue où il suffit de fournir les principales informations demandées. Parmi les classes d'objets disponibles, on note les classes déjà rencontrées, soit les **OleDBConnection**, les **OleDBCommand**. les **OleDbDataAdapter**, les **DataSet** et les **Dataview** avec en plus la classe **DataGrid** qui permet l'affichage intégral d'un jeu d'enregistrements.

Bien que tous ces objets puissent être créés lors de l'exécution du programme, le fait de pouvoir les définir en mode *Design* facilite la tâche par l'utilisation des fenêtres de propriétés et des boîtes de dialogue des assistants.

Schéma général

On peut schématiser ainsi la chaîne des relations entre les différents objets entre les données de la base et la grille d'affichage sur la feuille :

Connexion ↔ **DataAdapter** ↔ **DataSet** ↔ **DataGrid**

En plaçant un **DataAdapter** sur la feuille, un assistant est lancé qui pourra définir ET la connexion (chaîne de connexion), ET la source des données (chaîne SQL). Ensuite il n'y aura plus qu'à ajouter sur la feuille, le **DataSet** et la **DataGrid** et d'établir les liens entre eux.

Classe DataGrid

Le point central de l'opération est certes l'utilisation du contrôle **DataGrid** car celui-ci peut afficher directement un jeu de caractère quelconque. Mieux, comme il s'agit d'un contrôle «dynamique», il suffit de lui affecter un jeu d'enregistrement pour qu'il s'adapte à celui-ci. En effet, le nombre, le titre et la largeur des colonnes est défini en fonction du jeu d'enregistrement qui l'alimente. Mieux, si on lui affecte plus d'un jeu d'enregistrements, le DataGrid se retrouve muni d'un bloc d'onglets permettant de choisir laquelle des tables correspondantes sera affichée.

L'information ne circule pas directement du DataGrid vers la base : elle doit transiter par une Table d'un DataSet. La propriété DataSource du DataGrid permettra donc de faire le lien avec le DataSet utilisé et avec la table qui s'y trouve.

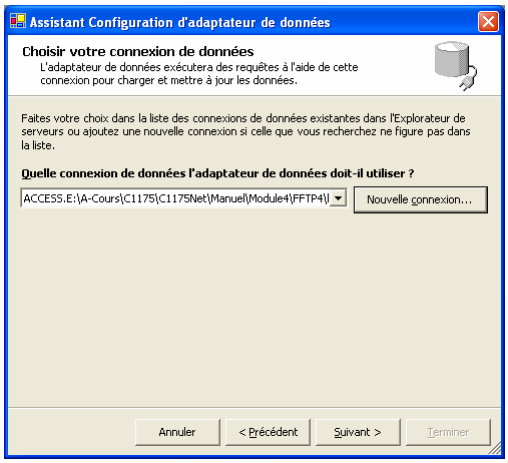
Selon que le **DataAdapter** qui lui sera associé pourra ou non modifier les enregistrements, les informations du DataGrid pourront ou non être mises à jour. Dans ce dernier cas, il est préférable de mettre sa propriété «ReadOnly» à True pour éviter que l'utilisateur ait la fausse impression de fasse des changements qui ne seront jamais transmis à sa base de données.

Enfin, la faiblesse actuelle du contrôle est la difficulté d'en modifier le format, ce qui sera sûrement amélioré lors d'une prochaine version de VB.NET. Notez cependant qu'une fois les données affichées, l'utilisateur peut gérer la largeur des colonnes et trier les informations sur chacune de celles-ci, au choix, en cliquant sur la tête de colonne.

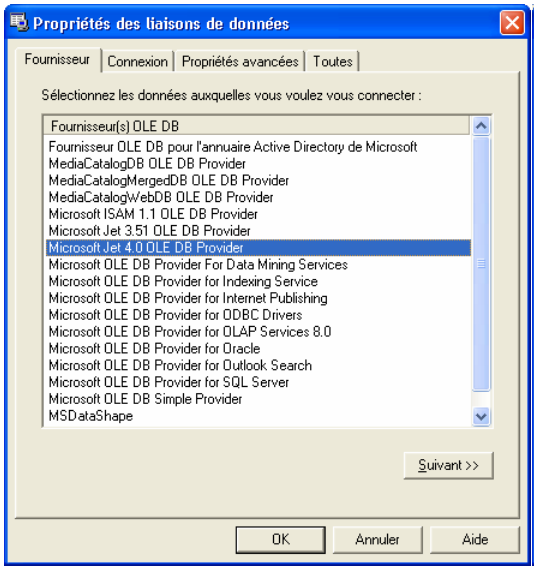
Classe OleDbConnection

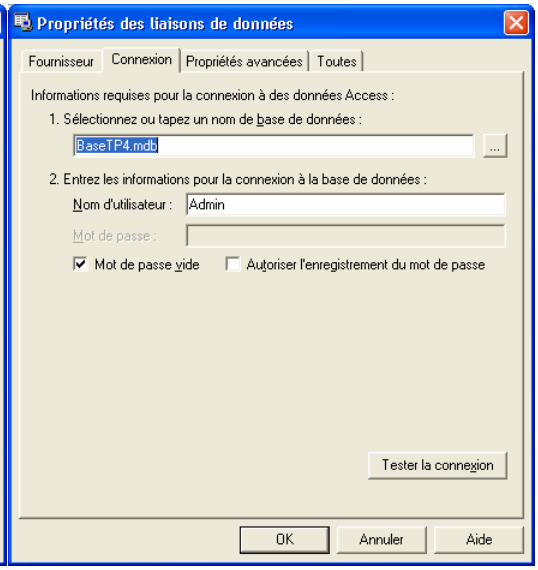
Bien que l'objet de classe **OleDbConnection** existe dans la boîte d'outils, on l'utilisera rarement puisque l'assistant **DataAdapter** permettra soit d'utiliser une connexion déjà établie, soit d'en créer une nouvelle. Notez qu'une même connexion peut servir à plusieurs «DataAdapter».

Classe OleDbDataAdapter



En déposant sur la feuille un contrôle **OleDbDataAdapter** (que ce soit par glissement ou par double-clic), on lance un «Assistant Configuration d'adaptateur de données» qui permettra de créer la liaison avec les données de façon interactive. De fait, l'assistant permettra de désigner le fournisseur de la base, de pointer vers le fichier (base de données) et donc de créer coup sur coup la chaîne de connexion ET la chaîne SQL, le tout de manière interactive.





La première étape de l'assistant consiste à désigner soit une connexion existante (déjà définie) ou d'en créer une nouvelle. Dans ce dernier cas, il fonctionne tout comme lors de la création d'un lien dynamique de données (voir plus haut, les fichiers «UDL») en permettant de choisir le fournisseur et la base de données.

L'assistant créera en même temps l'objet de connexion et le *DataAdapter*. Bien sûr, ces contrôles ne seront pas visibles sur la feuille elle-même mais seront plutôt placés sur «l'annexe» au bas de la feuille.

Il faudra utiliser autant de *DataAdapter* qu'il y a de tables à pourvoir. Lors de la création de chacun d'eux, la chaîne SQL définie («SELECT ... ») permettra l'affichage de ces données mais non leur mise à jour. L'assistant créera (s'il le peut) les autres chaînes SQL nécessaires à l'ajout, la suppression et la modification des enregistrements («Insert ... », «Delete ...» et «Update ...»). Dans le cas du TP, la première grille sera mise en mode de lecture seule. Elle permettra l'affichage (seul) des données provenant des trois tables de la base de données grâce à la requête Access (incluse dans la base) «ListeGlobale» qui pige dans les trois tables. Cependant le DataAdapter ne pourra pas construire les autres commandes SQL nécessaires à la modification.

Dans le cas de la deuxième grille, on la connectera à un *DataSet* contenant trois tables indépendantes placées dans un même DataSet. par chacun des trois DataAdapter associés à chacune d'elles.

Classe DataSet

Pour chaque grille, on ajoutera un contrôle **DataSet** qui pourra recevoir le jeu d'enregistrement défini par la chaîne SQL de chaque *DataAdapter*. Une fois la grille associée au DataSet, il n'y aura plus qu'à commander le déplacement des informations de la base au DataSet, donc vers la grille, et de la grille (via le *DataSet*) vers la base de données.

Attention cependant! Comme ADO.NET est basé sur le traitement indépendant des données en mémoire, il faudra procéder au transfert PAR PROGRAMMATION. Pour ce faire, le *DataSet* possède une méthode Fill(*UnDataSet*, "*nom_d'une_table*") qui ouvre la connexion, lance l'appel à la requête SQL «SELECT... », place le jeu d'enregistrements dans la table nommée et referme la collection. Comme le DataGrid est lié à la table, les informations y sont directement accessibles. On utilise cette méthode lors du chargement de la feuille ou bien en cours d'exécution (via un bouton de commande) chaque fois que l'on désire recharger les informations de la base (par exemple si une autre grille y a placé des mises à jour).

En parallèle, si le *DataAdapter* a pu générer les requêtes SQL adéquates, lors la méthode Update*(UnDataSet,* "*nom_d'une_table*") ou Update*(UnDataSet)* mettra à jour les informations contenue dans une table ou dans toutes les tables du *Dataset* au moyen de ces chaînes SQL. Le DataAdapter ne transférera bien sûr que les enregistrement modifiés et la méthode retournera une valeur entière représentant le nombre d'enregistrements touchés (créés, modifiés ou supprimés). Ici aussi, la connexion sera ouverte et refermée automatiquement.

On utilisera cette méthode soit à la fermeture de l'application, soit au besoin au moyen d'un bouton de commande.