Table des Matières

Introduction

Visual Basic de Microsoft est l'outil le plus rapide et le plus facile à utiliser pour créer des applications Microsoft Windows®. Que vous soyez un programmeur professionnel ou que vous découvriez la programmation sous Windows, Visual Basic vous offre une gamme complète d'outils qui simplifient et accélèrent le développement d'applications.

De quoi se compose Visual Basic ?

Le mot **«Visual»** fait référence à la méthode utilisée pour créer l'interface graphique utilisateur (GUI, *Graphical User Interface*). Au lieu de rédiger de multiples lignes de code pour décrire l'apparence et l'emplacement des éléments d'interface, il vous suffit d'ajouter et de placer des objets prédéfinis à l'endroit adéquat sur l'écran. La création et la gestion de l'interface graphique sont très simple et ne demandent pas des compétences spéciales et si vous avez déjà utilisé des programmes de dessin tels que Paint, vous disposez de la plupart des compétences requises pour créer une interface utilisateur performante.

Le mot **« Basic »** fait référence au langage **BASIC** (*Beginners All-Purpose Symbolic Instruction Code*), langage le plus utilisé par les programmeurs depuis les débuts de l'informatique. **Visual Basic** constitue une évolution par rapport au langage **BASIC** initial et comporte aujourd'hui plusieurs centaines d'instructions, de fonctions et de mots clés, dont un grand nombre font directement référence à l'interface graphique utilisateur (GUI) de Windows. Si vous débutez, vous serez en mesure de créer des applications très utiles en n'apprenant que quelques mots clés ; et si vous êtes un programmeur professionnel, la puissance de ce langage vous permettra de développer tout ce qu'il est possible de développer avec tout autre langage de programmation Windows.

Que vous ayez l'intention de créer un petit utilitaire pour vous-même ou votre groupe de travail, un système à l'échelle de l'entreprise ou même des applications partagées internationales sur Internet, **Visual Basic** met à disposition tous les outils nécessaires.

Visual Basic et les autres Langages traditionnels

Pour bien comprendre le processus de développement d'une application, il est utile de cerner certains des concepts essentiels de **Visual Basic**. Comme il s'agit d'un langage de développement Windows, vous devez être familiarisé avec l'environnement Windows. Si vous n'avez jamais programmé sous Windows, sachez qu'il existe des différences fondamentales entre la programmation pour Windows et la programmation pour d'autres environnements (MS-Dos …).

Système d'exploitation Windows

Une version simplifiée de la mécanique de fonctionnement de Windows s'articulerait autour de trois concepts essentiels : les fenêtres - les événements - les messages.

a- Les Fenêtres :

Considérez qu'une fenêtre est simplement une zone rectangulaire dotée de ses propres limites. Sans doute, connaissez-vous déjà plusieurs types de fenêtres : une fenêtre Explorateur dans Windows 95, une fenêtre de document dans un programme de traitement de texte, ou encore une boîte de dialogue qui apparaît pour vous rappeler un rendez-vous important. S'il s'agit là des exemples les plus courants, il existe bien d'autres types de fenêtres. Un bouton de commande est une fenêtre. Les icônes, les zones de texte, les boutons d'option et les barres de menus constituent tous des fenêtres.

b- Les Evénements :

Le système d'Exploitation Windows surveille en permanence chacune de ces fenêtres de façon à déceler le moindre événement ou signe d'activité. Les événements peuvent être engendrés par des actions de l'utilisateur (notamment lorsque celui-ci clique un bouton de la souris ou appuie sur une touche), par un contrôle programmé, voire même par des actions d'une autre fenêtre.

c- Les Messages :

Chaque fois qu'un événement (Clique – Double Clique …) survient, un message est envoyé au système d'exploitation. Celui-ci traite le message et le diffuse aux autres fenêtres. Chacune d'elles peut alors exécuter l'action appropriée de la manière prévue pour ce type de message (notamment, se redessiner si elle n'est plus recouverte par une autre fenêtre).

Comme vous pouvez l'imaginer, il n'est pas simple de faire face à toutes les combinaisons possibles de fenêtres, d'événements et de messages. Heureusement, Visual Basic vous épargne la gestion de tous les messages de bas niveau. La plupart de ceux-ci sont automatiquement gérés par Visual Basic tandis que d'autres sont mis à disposition sous forme de procédures Event afin de vous faciliter la tâche. Vous pouvez aussi écrire rapidement des applications puissantes sans vous préoccuper de détails inutiles.

Description du modèle événementiel

Dans les **Applications Procédurales**, c'est l'application elle-même, et non un événement, qui contrôle les parties du code qui sont exécutées, ainsi que leur ordre d'exécution. Celle-ci commence à la première ligne de code et suit un chemin défini dans l'application, appelant les procédures au fur et à mesure des besoins.

Dans une **application événementielle**, le code ne suit pas un chemin prédéterminé. Différentes sections du code sont exécutées en réaction aux événements. Ceux-ci peuvent être déclenchés par des actions de l'utilisateur, par des messages provenant du système ou d'autres applications, voire même par l'application proprement dite. L'ordre de ces événements détermine l'ordre d'exécution du code. Le chemin parcouru dans le code de l'application est donc différent lors de chaque exécution du programme.

Comme il est impossible de prévoir l'ordre des événements, votre code doit émettre certaines hypothèses quant à « l'état du système » au moment de son exécution. Lorsque vous élaborez des hypothèses (par exemple quand vous supposez qu'un champ de saisie doit contenir une valeur avant l'exécution de la procédure chargée de traiter cette valeur), vous devez structurer votre application de telle sorte que cette hypothèse soit toujours vérifiée (par exemple en désactivant le bouton de commande qui démarre la procédure aussi longtemps que le champ de saisie ne contient pas une valeur).

Votre code peut également déclencher des événements pendant l'exécution. Par exemple, la modification par programmation du contenu d'une zone de texte déclenche l'événement Change qui lui est associé et donc l'exécution du code éventuellement contenu dans cet événement. Si vous avez estimé que cet événement ne serait déclenché que par dialogue avec l'utilisateur, vous risquez d'être confronté à des résultats inattendus. C'est pour cette raison qu'il est important de bien comprendre le modèle événementiel et de le garder toujours à l'esprit tout au long de la phase de création d'une application.

Développement interactif :

Le processus traditionnel de développement des applications peut être divisé en trois étapes distinctes : l'écriture, la compilation et la vérification du code. Contrairement aux langages traditionnels, Visual Basic adopte une approche interactive dans laquelle disparaissent les distinctions entre ces trois étapes.

Dans la plupart des langages, une erreur commise dans l'écriture du code n'apparaît qu'au moment de la compilation de l'application. Vous devez ensuite rechercher et corriger l'erreur avant de recommencer le cycle de compilation, et répéter ce processus pour chaque erreur. Visual Basic procède de manière totalement différente : il interprète le code au fur et à mesure de sa saisie, interceptant et signalant immédiatement la plupart des erreurs de syntaxe et des fautes d'orthographe. C'est un peu comme si un expert était assis à vos côtés, vérifiant chaque ligne de code que vous introduisez.

Outre la détection immédiate des erreurs, Visual Basic compile également partiellement le code au moment de sa saisie. Lorsque vous êtes prêt à exécuter et vérifier votre application, la compilation est déjà presque pratiquement terminée. Si le compilateur découvre une erreur, il l'affiche en surbrillance dans votre code. Vous pouvez alors corriger l'erreur et poursuivre la compilation sans devoir recommencer depuis le début.

En raison de la nature interactive de Visual Basic, vous exécuterez fréquemment votre application tout au fil de son développement. Vous pourrez ainsi tester les effets de votre code alors même que vous l'écrivez, plutôt que d'attendre sa compilation ultérieure.

Concepts Fondamentaux

Avant de commencer votre projet, Vous devez vous familiariser avec certains concepts / éléments propres au **Visual Basic**.

1- Définition d'un Projet Visual Basic

Pour créer une application avec Visual Basic, vous utilisez des projets. Un **projet** est un ensemble de fichiers qui permet de concevoir une application.

Lorsque vous créez une application, vous créez généralement de nouvelles feuilles, mais vous pouvez aussi réutiliser ou modifier des feuilles que vous avez déjà créées pour de précédents projets.

2- Définition d'une feuille

Les feuilles sont des objets qui possèdent des propriétés définissant leur apparence, des méthodes définissant leur comportement et des événements définissant l'interaction avec l'utilisateur. En définissant les propriétés d'une feuille et en écrivant le code Visual Basic pour répondre à ses événements, vous personnalisez l'objet en fonction des exigences de votre application.

La première étape de création d'une application à l'aide de Visual Basic consiste à créer l'interface, c'est-à-dire la partie visuelle de l'application avec laquelle l'utilisateur dialogue. Les feuilles et les contrôles sont les éléments de base qui permettent de créer l'interface ; ce sont les objets que vous utilisez pour construire votre application.

3- Définition des Contrôles

Les contrôles sont des objets qui sont inclus (insérés - disposés) dans des objets feuille. Chaque type de contrôle possède son propre jeu de propriétés, de méthodes et d'événements qui permettent de l'adapter à un usage particulier. Parmi les contrôles que vous pouvez utiliser dans vos applications, certains conviennent mieux pour saisir ou afficher du texte. D'autres permettent d'accéder à des BD pour traiter les données etc.

Une fois que vous avez assemblée tous les composants d'un projet et écrit le code, vous devez compiler le projet afin de créer un fichier exécutable.

L'Interface de Développement

L'environnement de développement Visual Basic comprend les éléments suivants :

1- Barre de menus

Affiche les commandes qui vous permettent d'utiliser Visual Basic. Outre les menus habituels, à savoir Fichier, Edition, Affichage, Fenêtre et **?** (Aide), des menus permettent d'accéder à des fonctions spécifiques nécessaires à la programmation, notamment les menus Projet, Exécution, Format et Débogage.

2- Barres d'outils

Permettent d'accéder instantanément à la plupart des commandes les plus courantes de l'environnement de programmation. Il vous suffit de cliquer une fois sur un bouton de la barre d'outils pour exécuter l'action qu'il représente. Par défaut, la barre d'outils Standard apparaît lorsque vous démarrez Visual Basic. Il existe d'autres barres d'outils pour la modification, la création de feuilles et le débogage. Elles peuvent être activées ou désactivées à l'aide de la commande Barres d'outils du menu Affichage.

3- Boîte à outils:

Fournit un ensemble d'outils nécessaires au moment de la création pour disposer les contrôles sur une feuille. Outre la disposition par défaut de la boîte à outils, vous pouvez ajouter d'autres contrôles à cette boîte selon le besoin.

4- Fenêtre Explorateur de projets :

Énumère les feuilles et les modules contenus dans votre projet en cours.

5- Fenêtre Propriétés :

Enumère les paramètres des propriétés de la feuille ou du contrôle sélectionné. Une propriété définit une caractéristique d'un objet, notamment sa taille, sa légende ou sa couleur.

6- Fenêtre de conception de feuille :

Fait office de fenêtre personnalisable pour la création de l'interface de votre application. Vous ajoutez des contrôles, des graphismes et des images à une feuille de façon à ce qu'elle prenne l'apparence souhaitée. Chaque feuille de votre application possède sa propre fenêtre de conception de feuille.

7- Fenêtre Présentation des feuilles :

La fenêtre Présentation feuille vous permet de positionner les feuilles dans votre application grâce à une petite représentation graphique de l'écran.

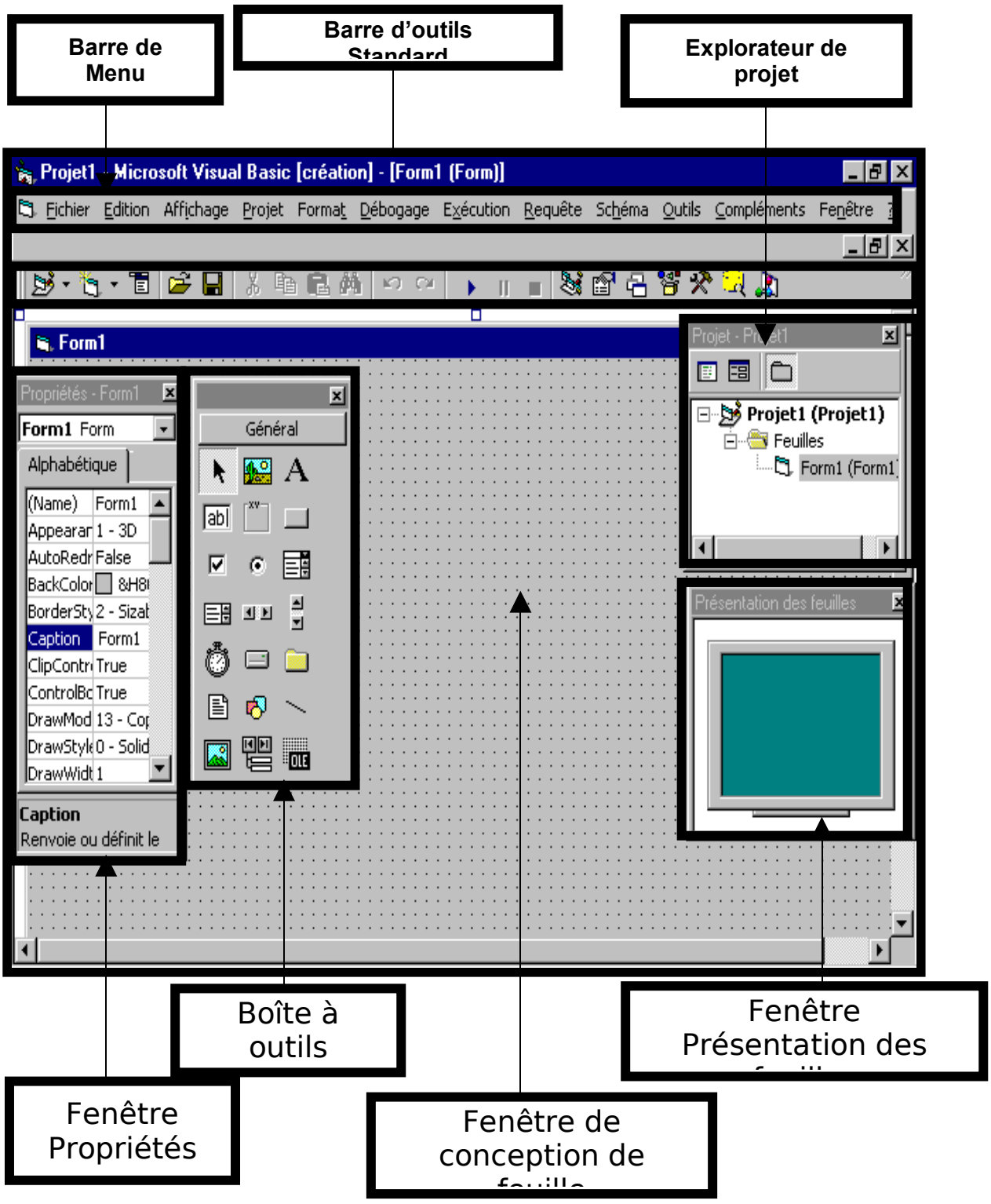


Figure 1 : Interface de développement VB

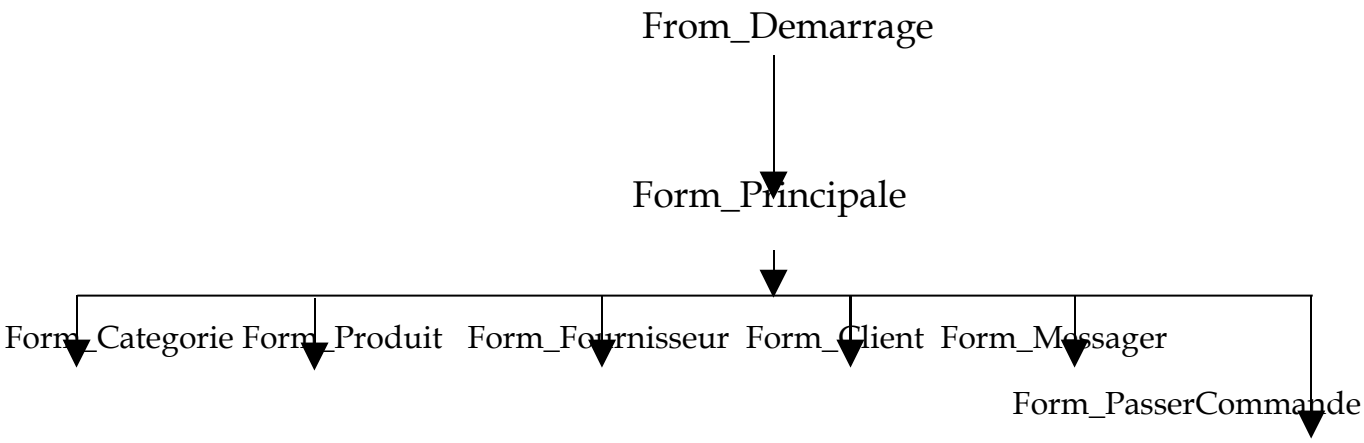
L'APPLICATION GESTION DE STOCK

Gestion de Stock est une application qu'on va devoir réaliser au cours des prochains TP. Cette application comme l'indique son nom devra permettre la Gestion des Produits/ commandes/ clients etc. il est bien évident qu'elle sera réalisée sous VB mais elle devrait communiquer avec une base de données access.

Avant d'entamer les TP, on devrait avoir une idée sur notre application (Interface – Base de données).

L'Interface : La hiérarchie des FORMS

L'interface de l'application sera constituée de x Forms qui auront la hiérarchie suivante :



La Base de données : La Structure de la Base de données

La base de données Stocks est réalisée sous Access 97. Elle contient 7 Tables, voici la Structure de ces tables :

La table Catégorie:

Code	Désignation	Type	Taille
CodCat	Code Catégorie	Entier	5
DesCat	Désignation Catégorie	Mémo	
DscCat	Description Catégorie	Mémo	

La table Fournisseur:

Code	Désignation	Type	Taille
CodFrs	Code Fournisseur	Texte	5
DesFrs	Désignation Fournisseur	Mémo	
AdrFrs	Adresse Fournisseur	Texte	250
VilFrs	Ville	Texte	100
PysFrs	Pays	Texte	100
TelFrs	Téléphone	Texte	20
FaxFrs	Fax	Texte	20

La table Client :

Code	Désignation	Type	Taille
CodClt	Code Client	Texte	5
DesClt	Désignation Client	Mémo	
AdrClt	Adresse Client	Texte	250
VilClt	Ville	Texte	100
PysClt	Pays	Texte	100
TelClt	Téléphone	Texte	20
FaxClt	Fax	Texte	20

La table Produit :

Code	Désignation	Type	Taille
RefPdt	Référence Produit	Entier	5
NomPdt	Nom Produit	Mémo	
CodFrs	Code Fournisseur	Entier	5
CodCat	Code Catégorie	Entier	5
QtePdt	Quantité	Entier	5
PUPdt	Prix Unitaire	Double	
QtCmdPdt	Quantité Commandée	Entier	5
QtSckPdt	Quantité en Stock	Entier	5

La table Commande :

Code	Désignation	Type	Taille
RefCmd	Référence Commande	Entier	5
CodClt	Code Client	Entier	5
DatCmd	Date Commande	Date	8
LivAva	Livré avant	Date	8
DatEnv	Date d'envoi	Date	8
DesCmd	Destination Commande	Texte	150
AdrLiv	Adresse Livraison	Texte	250
vilLiv	Ville Livraison	Texte	100
PysLiv	Pays Livraison	Texte	100

La table DetailCommande :

Code	Désignation	Type	Taille
RefCmd	Référence Commande	Entier	5
CodClt	Code Client	Texte	5
QteCmdPdt	Quantité Commandée	Entier	10
PUPdt	Ville Livraison	Double	10
PysLiv	Pays Livraison	Texte	100

Création de l'Application Stock

Etape1 :Création d'un nouveau Projet

Pour créer votre première application Visual Basic :

1- Cliquer sur **Nouveau** du menu **Fichier** ;

Dans la boite de dialogue « Nouveau Projet » qui apparaîtra à l'écran :

2- Sélectionner le type d'application, ici : EXE.Standard ;

3- Valider par **OK** ;

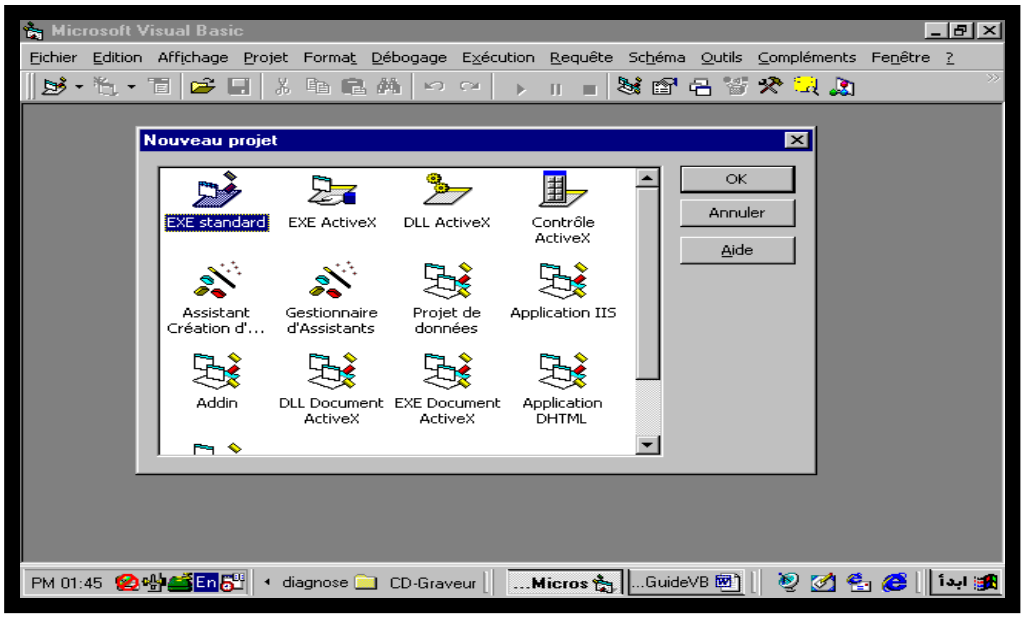


Figure 1 : La Boîte de Dialogue « Nouveau Projet »

Les Types de Projet :

Visual **B**asic vous propose un certain nombre de type d'application que vous pouvez élaborer sous VB, en voici quelques exemples :

EXE.Standard :

Pour créer des applications standards (exe : Projet de gestion) ;

Contrôles Actives X:

Création des contrôles actives X ;

Application DHTML:

Concevoir des applications Internet utilisant le DHTML ;

Maintenant que vous avez créer votre premier projet, vous remarquer, dans l'explorateur de projet, que la première feuille a été créée automatiquement et a le nom «Form1». Vous pouvez ajouter d'autres feuilles chaque fois que ça s'avère nécessaire.

TP1 : Feuille de démarrage

Pour le moment vous allez déterminer les propriétés de cette feuille « Form1 » et y insérer en suite des objets « Contrôles ».

Etape 1 : Modification des Propriétés de la Feuille

Vous allez utiliser la Fenêtre des Propriétés pour modifier l'aspect de la feuille. Vous remarquez qu'un grand nombre des propriétés affectent l'aspect physique de la feuille.

Voici une liste des propriétés qui figure dans la Fenêtre des Propriétés avec une explication décrivant l'impact de chacune sur la feuille (ou tout autre contrôle qu'on insérera après) :

Propriété	Définition
Name	Définit le nom qui désigne la feuille dans le code. Lorsque vous ajoutez pour la première fois une feuille à un projet, son nom par défaut est Form1, Form2, etc. Il est conseillé de définir la propriété Name en choisissant un nom plus significatif, par exemple « frmDemarrage » pour une feuille d'entrée de commandes.
Caption	Vous pouvez la modifier. Elle définit le texte figurant dans la barre de titre de la feuille.
Icon	L'icône qui s'affiche lorsque vous réduisez la feuille. Et qui apparaît à gauche du titre de la barre de titre
MaxButton Et MinButton	Déterminent quant à elles si la feuille peut être agrandie ou réduite.
BorderStyle	En modifiant cette propriété vous pouvez contrôler le redimensionement de la feuille
Height et With	Déterminent la taille initiale d'une feuille.
Left et Top	Détermine la position par rapport à l'angle supérieur gauche de l'écran.
WindowState	Vous pouvez définir pour démarrer la feuille sous forme agrandie, réduite ou normale.

Conseil

La meilleure façon de vous familiariser avec les nombreuses propriétés d'une feuille est de vous exercer à les utiliser. Modifiez quelques propriétés d'une feuille dans la fenêtre Propriétés, puis exécutez l'application pour voir le résultat. Pour obtenir des informations complémentaires sur une propriété, sélectionnez-la et appuyez sur F1 pour afficher l'aide contextuelle.

Travail à faire :

- Modifier la Propriété **Caption** en lui affectant le titre suivant : **« Gestion Stock »**
- Modifier la Propriété **BackColor** en choisissant une autre couleur
- Modifier la propriété **Heigt** = 4500 ;
- Modifier la propriété **With** = 6000 ;
- Modifier la propriété **Left** =1395 ;
- Modifier la propriété **Top**=1500 ;
- Modifier la Propriété **Name** = **Frm_Demarrage** ;

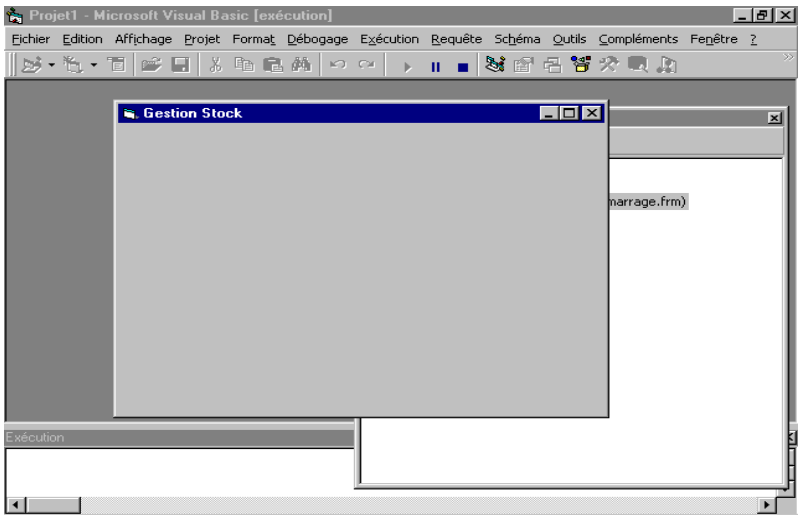
Exécuter pour voir le résultat.

L'Exécution : Vous pouvez exécuter votre programme en procédant de 3 manières :

1 - Cliquer sur F5 ;

2 - Cliquer sur le bouton de la barre d'outil ;

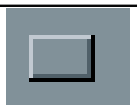
3 - Cliquer sur Exécuter Du Menu Exécution ;

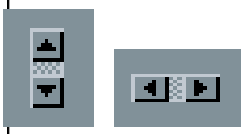
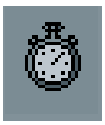


TP 1. Figure1 : Resultat de la 1ère Etape

Etape 2 : Ajout des contrôles « Objets » à la feuille

Dans La Boîte à outils vous trouverez un grand nombre de contrôle que vous pouvez insérer dans votre feuille :

Contrôle	Définition
	Ne vous fiez pas aux apparences. Le Pointeur n'est pas un contrôle. Il sert juste à déplacer et à redimensionner un contrôle placé sur une feuille.
	Le contrôle PictureBox sert à afficher une image en mode point, une icône ou un métafichier. Il sert aussi à regrouper des boutons radio. Remarquons que ce contrôle est le seul qui puisse être placé dans une feuille MDI. En effet, essayez de placer d'autres contrôles sur une feuille MDI, vous verrez que ça ne marche pas !
	Le contrôle Label sert à placer du texte qui ne peut être modifié ou effacé lors de l'exécution de l'application.
	Le contrôle Textbox sert à placer du texte qui peut être ou non modifié par l'utilisateur lors de l'exécution de l'application.
	Le contrôle Frame sert à regrouper plusieurs contrôles(checkbox, optionbutton,...) dans un cadre avec un titre que vous pouvez modifier dans ses propriétés.
	Le contrôle CommandButton sert à afficher un bouton de commande qui lorsque l'on clique dessus, la portion de code écrite dans sa procédure est exécutée.
	Le contrôle Checkbox sert à placer une case à cocher. En plaçant plusieurs cases à cocher dans une application, l'utilisateur aura le choix entre plusieurs options. Selon les options sélectionnées, une portion de code sera exécutée.
	Le contrôle OptionButton s'utilise comme un contrôle CheckBox. Contrairement aux cases à cocher, une seule option peut être sélectionnée parmi un groupe d'options composées de boutons radio.
	Le contrôle ComboBox laisse à l'utilisateur le choix de sélectionner un élément parmi d'autres dans une liste.

	Le contrôle ListBox s'utilise comme le contrôle ComboBox sauf que dans la liste d'option, plusieurs options sont affichées simultanément.
	Les barres de défilement permettent de lire la suite du contenu d'une fenêtre trop large ou trop longue pour être affichée en une seule fois.
	Le contrôle Timer permet de générer un événement à intervalle régulier, par exemple dans une horloge où toutes les minutes, l'aiguille des minutes bouge. Notons que le contrôle Timer n'apparaît pas pendant l'exécution d'une application.
	Le contrôle Shape permet de dessiner des figures dans une feuille (rectangle, carré, cercle). Le type de forme est choisi dans la propriété Shape du contrôle.
	Le contrôle Line permet de tracer des lignes dans une feuille. La propriété BorderStyle permet de choisir le type de ligne que vous désirez : continu, invisible, pointillé...
	Le contrôle Image permet tout comme le contrôle PictureBox d'insérer des images en mode point, icône ou métafichier. Cependant, il requiert moins de ressource et donc diminue la taille de votre application.
	Le contrôle Data permet d'accéder aux données provenant de bases de données.

1- Ajout d'un contrôle Label

- Dans la boîte à outils cliquer sur le contrôle **Label** ;
- Disposer le contrôle sur la feuille ;
- Dans la fenêtre de propriétés, modifier le contenu de la propriété **Caption** = « GESTION DE STOCK » ;
- Modifier la propriété **Font** en choisissant la police « Times New Roman » et la taille 28 ;
- Modifier la propriété **BackStyle** =transparent ;
- Modifier la Propriété **ForeColor** en choisissant la couleur « Bleu »;

2- Ajout d'un contrôle Bouton de Commande :

- Dans la boîte à outils cliquer sur le contrôle Bouton de commande ;
- Disposer le contrôle sur la feuille ;
- Dans la fenêtre de propriétés, modifier la propriété **Name** = « Cmd_Quitter » ;

- Modifier le contenu de la propriété **Caption** = « Quitter » ;
- Modifier la propriété **Font** en choisissant la police « Times New Roman » et la taille 11 et la style **« Gras »** ;

3- Ajout d'un contrôle Image :

Nous allons maintenant ajouter une image à notre feuille pour qu'elle ail une apparence un peu plus attrayante

- Dans la boîte à outils cliquer sur le contrôle Image ;
- Disposer le contrôle sur la feuille ;
- Dans la fenêtre de propriétés, modifier la propriété picture en choisissant l'image **« Bd06566_ »** ;
- Modifiez la propriété **Stretch** en choisissant la valeur « True » pour que l'image soit redimensionnable ;

TP 1. Figure2 : Resultat de la 2ème Etape

Etape 3 : Ajout du Code

Vous vous souvenez du bouton « Quitter » et vous remarquer que quand vous cliquez là dessus en mode exécution rien ne se passe !!

Un bouton n'a aucune utilité s'il n'y a pas des instructions « code » derrière pour lui dicter son comportement vis à vis des événements effectuer par l'utilisateur.

Les instructions sont souvent écrites au sein d'une ou plusieurs procédure(s) et se sont ces procédures qui génère les actions.

A présent, analysons la structure d'une procédure. La syntaxe d'écriture d'une procédure est la suivante :

```
[Public / Private]  Sub Nom_procédure (arguments)
```

```
[Déclarations]
[Instructtions]
[Instructions]
End Sub
```

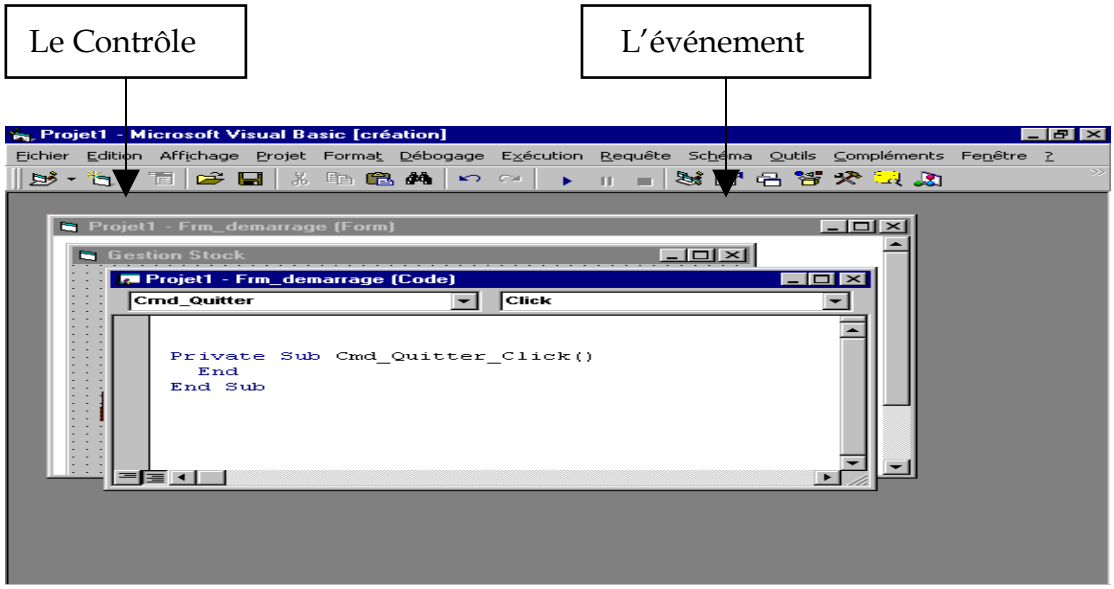
Le mot Sub peut ou non être précédé des options suivantes :

1. Les mots **Public** ou **Private** définissent les limites de la procédure.
   - Avec **Public**, la procédure peut être appelée depuis n'importe quelle instruction de l'application.
   - Avec **private**, la procédure ne peut être appelée qu'à l'intérieur de la même feuille ou du même module.
2. Vous devez déclarer en début de procédures vos variables et constantes si vous ne l'avez pas fait dans la partie "Général" de la liste déroulante Objet qui se trouve en haut, à gauche de l'éditeur de code. Remarquez que si vous déclarez vos variables à l'intérieur de la procédure, sa portée sera limitée qu'à cette procédure;
3. **End Sub** ferme la procédure.

Pour ouvrir la fenêtre dans laquelle on saisit les lignes de code lier à un contrôle, il vous suffit de double cliquer sur l'objet concerné ici : le bouton « Quitter » ;

Travail à faire

Dans l'événement **Click** du bouton de commande « Cmd_Quitter » saisissez **End;**



Private Sub : Déclare la procédure Cmd_Quitter_Click() ;

Cmd_Quitter_Click() : le nom de la procédure, c'est la concaténation du nom du Contrôle et de l'événement ;

End Sub : Annonce la fin de la Procédure ;

Maintenant exécutez l'application et cliquez sur le bouton quitter pour voir le résultat.

Enregistrer Le Projet :

1- Cliquer sur Enregistrer le Projet du menu Fichier ;

Dans la boite de dialogue « Enregistrer Fichier » qui apparaîtra à l'écran :

2- Enregistrer la feuille Frm_Demarrage ;

3- Enregistrer le projet Stock ;

Fin du TP1

Exercice1 : Création de la feuille à propos

Comment ajouter une nouvelle feuille au projet ?

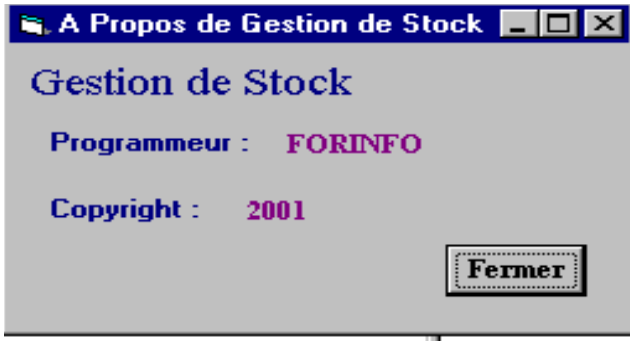
Du menu Projet cliquez sur Ajouter une Feuille ;

Dans la boîte de dialogue « Ajouter une Feuille », Choisissez Form ;

Vous allez Ajouter une nouvelle feuille à l'application et vous allez suivre les mêmes étapes que le TP1 pour la mettre en forme et lui ajouter du Code.

Le nom de la feuille sera Frm_Propos ;

La feuille **Frm_Propos** aura, à la fin, l'aspect suivant :



TP2 : CREATION DE LA FEUILLE PRINCIPALE

Dans ce TP nous allons procéder à la création de la feuille principale qui sera la fenêtre parent. Cette feuille sera du type MDIFORM.

1- Etape1 : Création de la MDIForm :

Définition d'une MdiForm

Tout d'abord, définissons ce que c'est qu'une MdiForm. Vous connaissez sûrement le logiciel de traitement de texte WORD de Microsoft. C'est une application composée d'une fenêtre principale(appelé aussi fenêtre parent) et d'une ou plusieurs fenêtres de documents(appelés aussi fenêtre fille). MDIFORM sont en général pourvu d'un système de menu, d'une barre d'outils et d'une barre d'état. Elles permettent de visualiser plusieurs documents en même temps.

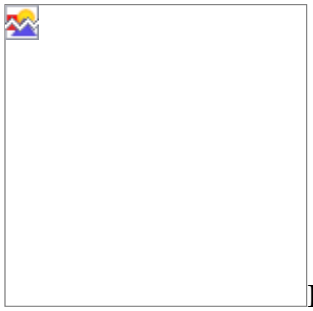
Travail à faire :

- Ajouter une Feuille Mdi en cliquant sur **Ajouter une feuille Mdi** du menu **Projet**;
- Modifier les Propriétés de la feuille Mdi :
  - La propriété Name = « Frm_Principale » ;
  - La propriété Caption = « Gestion de stock » ;
  - La Propriété WinsowState = « Maximised » ;

Etape2 : Création d'un système de Menu :

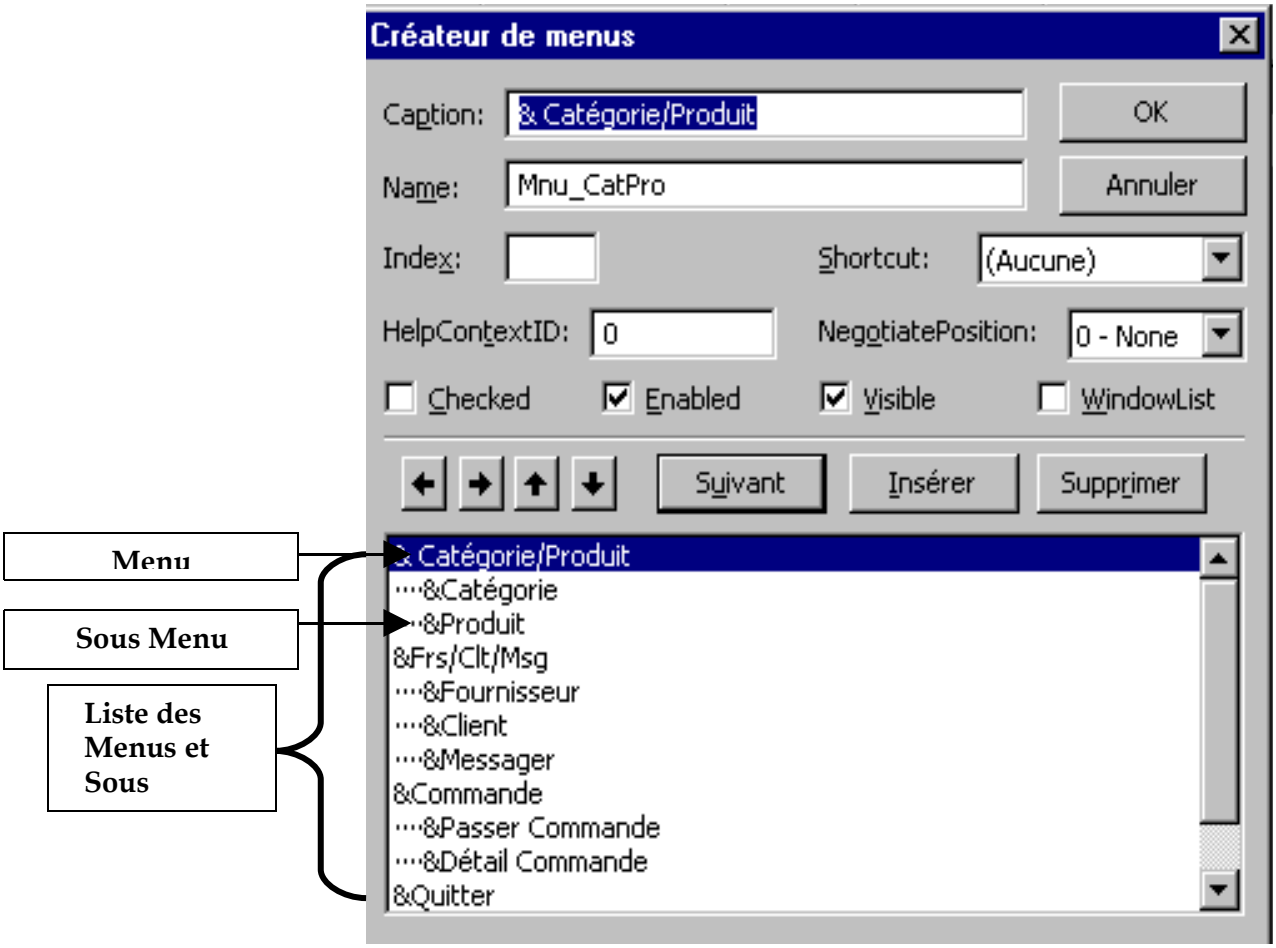
Définition d'une MdiForm

Un système de menus est composé de plusieurs menus qui eux-mêmes peuvent être composés de plusieurs sous-menus. La plupart des logiciels de Microsoft possèdent un système de menu.



Pour créer un système de menus, allez dans le menu "Outils" du système de menu de Visual Basic et choisissez la commande "Créateur de menus" ou pour aller plus vite, cliquez sur la 3ème image de la barre d'outils.

La boîte de dialogue « **Créateur de Menu** » s'affiche.



Travail à faire :

- Ajouter à la Feuille un **Système de Menu** ;

Une fois, c'est terminé, pour la création de menu. Il ne reste plus qu'à affecter à chaque commande de menu une action spécifique. Dans ce TP nous allons nous limiter à la création du code associé aux deux menus : Quitter et A propos ; pour les autres Commandes de menu on leur affectera leurs programmes ultérieurement.

- Ajouter le code associé au menu A Propos ;

Le menu A propos sevira à afficher la feuille A propos.

La Syntaxe qui nous permet d'afficher une feuille est la suivante :

Feuille.Show

Feuille : est le nom de l'objet feuille, c'est la valeur donnèes à la propriété name

Show : est l'instruction qui Charge la feuille.

```
Private Sub Mnu_Propos_Click()
    Frm_Propos.Show
```

```
End Sub
```

- Ajouter le code associé au menu Quitter

  Le menu Quitter devra permettre de sortir de l'application comme le Bouton quitter de la Feuille de demarrage ;

- Double Cliquez sur Le Menu quitter ;

- Dans la fenêtre du code écrivez le code adéquat ;

Exécuter le programme

Etape3 : Ajout des boites de dialogue :

a- Définition des boîtes de dialogue :

Les boîtes de dialogue des applications Windows permettent :

- D'inviter l'utilisateur à entrer les données que l'application requiert pour poursuivre son exécution.
- D'afficher des informations destinées à l'utilisateur.

La meilleure façon d'ajouter une boîte de dialogue à votre application est d'utiliser une boîte de dialogue prédéfinie car vous n'avez pas à vous préoccuper de sa création, de son chargement ou de son affichage. Toutefois, vous disposez de peu de possibilités pour intervenir sur leur aspect.

Voiçi la liste des fonctions que vous pouvez utiliser pour ajouter des boîtes de dialogue prédéfinies dans votre application Visual Basic.

Nom de la Fonction	Rôle de la Fonction
InputBox	Invite l'utilisateur à entrer des données
MsgBox	Afficher un message dans une boîte de dialogue et renvoyer une valeur indiquant le bouton de commande sur lequel l'utilisateur a cliqué.

b - LA FONCTION MsgBox :

Utilisez la fonction **MsgBox** pour communiquer avec l'utilisateur, assurer une interactivité, et obtenir des réponses de type oui/non de la part des utilisateurs en affichant de courts messages dans une boîte de dialogue (messages d'erreur, d'avertissement ou d'alerte). Après avoir lu le message, l'utilisateur clique sur un bouton pour fermer la boîte de dialogue.

Cette fonction permet d'affiche un message dans une boîte de dialogue qui attend que l'utilisateur clique sur un bouton, puis renvoie une valeur de type **Integer** qui indique le bouton choisi par l'utilisateur.

La Syntaxe de la fonction MsgBox est la suivante :

MsgBox (Prompt, [bouton], [titre])

La syntaxe de la fonction **MsgBox** comprend les arguments suivants :

Element	Description
Prompt	Expression de chaîne affichée comme message dans la boîte de dialogue. Si l'argument ***prompt*** occupe plus d'une ligne, n'oubliez pas d'insérer un retour chariot (**Chr**(13)) ou un saut de ligne (**Chr**(10)) entre les lignes, ou une combinaison de caractères retour chariot-saut de ligne (**Chr**(13) **&** **Chr**(10)). **Exemple :** **Msgbox** « Attention ! » **& chr(13) &** « Ce Programme est protégé », VbInformation
Buttons	Facultatif. Expression qui indique le nombre et le type de boutons à afficher, le style d'icône à utiliser, ainsi que l'identité du bouton par défaut. Si l'argument ***buttons*** est omis, sa valeur par défaut est 0 càd : **VbOkOnly.**
Title	Facultatif. Expression de chaîne affichée dans la barre de titre de la boîte de dialogue. Si l'argument **title** est omis, le nom de l'application est placé dans la barre de titre.

Voici une liste des valeurs que prendre L'argument bouton :

Constant	Value	Description
vbOKOnly	0	Affiche le bouton **OK** uniquement.
vbOKCancel	1	Affiche les boutons **OK** et **Annuler**.
vbAbortRetryIgnore	2	Affiche le bouton **Abandonner**, **Réessayer** et **Ignorer**.
vbYesNoCancel	3	Affiche les boutons **Oui**, **Non** et **Annuler**.
vbYesNo	4	Affiche les boutons **Oui** et **Non**.
vbRetryCancel	5	Affiche les boutons **Réessayer** et **Annuler**.
vbCritical	16	Affiche l'icône **Message critique**.
vbQuestion	32	Affiche l'icône **Requête d'avertissement**.
vbExclamation	48	Affiche l'icône **Message d'avertissement**.
vbInformation	64	Affiche l'icône **Message**

		d'information.
vbDefaultButton1	0	Le premier bouton est le bouton par défaut.
vbDefaultButton2	256	Le deuxième bouton est le bouton par défaut.
vbDefaultButton3	512	Le troisième bouton est le bouton par défaut.
vbDefaultButton4	768	Le quatrième bouton est le bouton par défaut.
vbApplicationModal	0	Boîte de dialogue modale. L'utilisateur doit répondre au message affiché dans la zone de message avant de pouvoir continuer de travailler dans l'application en cours.
vbSystemModal	4096	Modal système. Toutes les applications sont interrompues jusqu'à ce que l'utilisateur réponde au message affiché dans la zone de message.
vbMsgBoxHelpButton	16384	Ajoute le bouton Aide à la zone de message.
VbMsgBoxSetForeground	65536	Indique la fenêtre de zone de message comme fenêtre de premier plan.
vbMsgBoxRight	524288	Le teste est aligné à droite.
vbMsgBoxRtlReading	1048576	Indique que le texte doit apparaître de droite à gauche sur les systèmes hébraïques et arabes.

Le premier groupe de valeurs (0 à 5) décrit le nombre et le type de boutons affichés dans la boîte de dialogue. Le deuxième groupe (16, 32, 48 et 64) décrit le style d'icône. Le troisième groupe (0, 256 et 512) définit le bouton par défaut. Enfin, le quatrième groupe (0 et 4 096) détermine la modalité de la zone de message. Au moment d'additionner ces nombres pour obtenir la valeur finale de l'argument ***buttons***, ne sélectionnez qu'un seul nombre dans chaque groupe.

Les valeurs renvoyées quand l'utilisateur répond à une boîte de dialogue « **Msgbox** » en cliquent sur l'un des boutons de la boîte sont :

Constante	Valeur	Description
vbOK	1	**OK**
vbCancel	2	**Annuler**

vbAbort	3	**Abandonner**
vbRetry	4	**Réessayer**
vbIgnore	5	**Ignorer**
vbYes	6	**Oui**

Remarques :

Si la boîte de dialogue est dotée d'un bouton Annuler, appuyer sur Échap équivaut à cliquer sur Annuler. Toutefois, aucune valeur n'est renvoyée tant que l'utilisateur n'a pas cliqué sur l'un des autres boutons.

Travail à faire :

- Améliorer le code du menu quitter en ajoutant le code suivant :

```
Private Sub Mnu_Quitter_Click()
    Msgbox « Fin de Programme »,vbOkonly,  « Gestion Stock »
    end
End Sub
```

- **Explication du programme :**

  Chaque fois que l'utilisateur clique sur « Quitter » le programme affiche le message avant de sortir de l'application.

- Reprendre le code du menu quitter et modifier-le :

```
Private Sub Mnu_Quitter_Click()
    Reponse= Msgbox (« Etes vous sur de vouloir quitter cette application ? » ,
                vbOkNo,  « Gestion Stock »)
    If  reponse=6  then
       End
    End if
End Sub
```

- **Explication du programme :**

  Chaque fois que l'utilisateur clique sur quitter, le programme affiche le message pour lui demander de confirmer sa décision. Si la réponse est affirmative càd : la valeur renvoyée est VbOk ou 6 alors le programme met fin à l'exécution du programme.

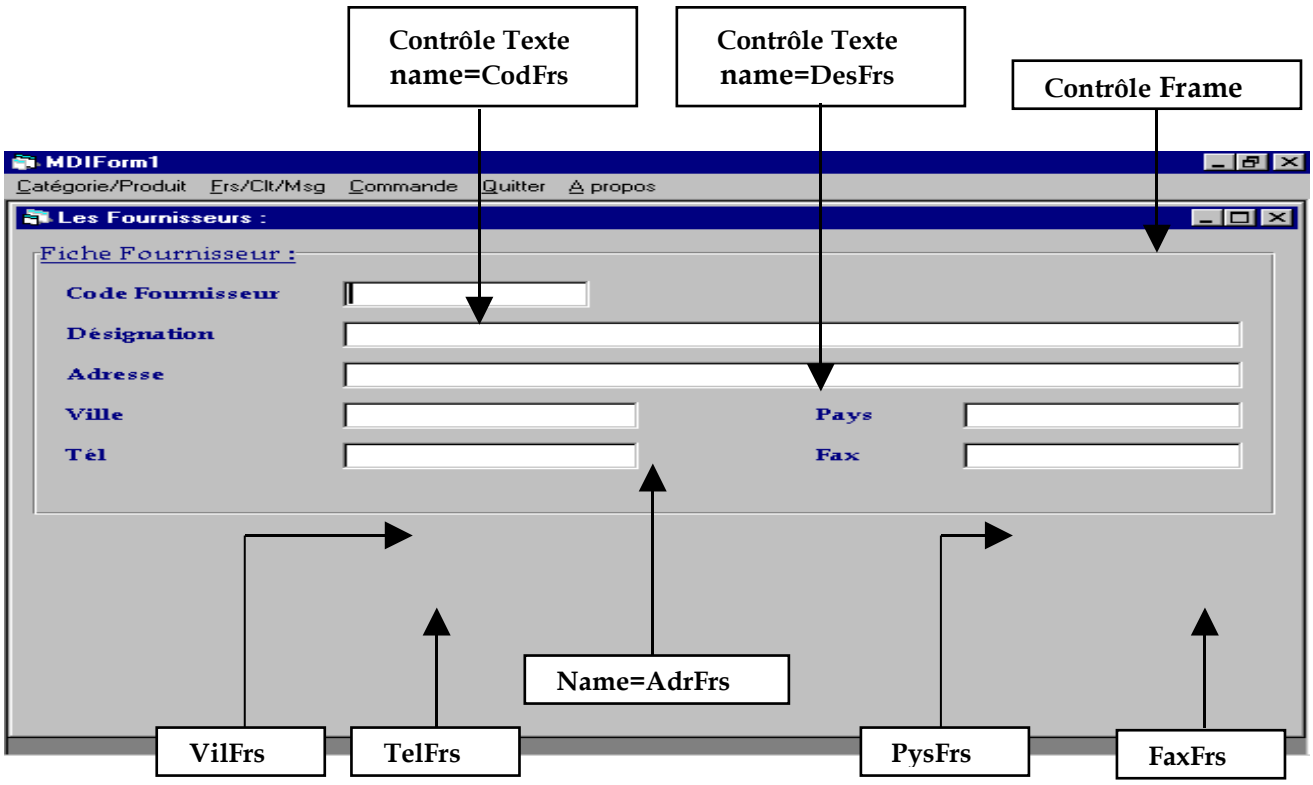
Fin TP2

TP3 : CREATION DE LA FEUILLE FOURNISSEUR

Au niveau de ce TP3 nous allons créer la form « Fournisseur» qui sera une form de consultation et modification des informations sur les fournisseurs.

1- Etape 1 : Préparer l'interface :

- Ajouter une nouvelle form à votre projet ;
- Donner à cette form le nom « Frm_Fournisseur » ;
- Réaliser l'interface suivante ;



Etape 2 : Ajout du Contrôl Data

Cette form devra nous permettre de consulter les données (les fournisseurs) de la table « Fournisseur » de la base de donnée « Bd_Stock ». Cette interface alors jouera le rôle d'intermédiaire entre l'utilisateur et la base de données et pour cela il faut lier les zones de textes à la table « Fournisseur ».

Pour lier des contrôles de la form à une base de données il faut disposer dans la form concerné un contrôle spécifique permettant d'effectuer le lien avec la BD c 'est le contrôle **« Data »**

a- Définition d'un Contrôle « Data »

Permet d'accéder aux données stockées dans les bases de données. Le contrôle Data vous permet de vous déplacer d'un enregistrement à l'autre, d'afficher et de manipuler les données à partir des enregistrements dans les contrôles dépendants.

L'élaboration d'une base de données se fait à l'aide du contrôle Data. Ainsi, vous pouvez accéder à pas mal de fichiers de données divers tels que ceux de Access, Excel, FoxPro ou tout fichier ASCII. En clair, le contrôle Data est indispensable dans la réalisation d'une gestion d'une base de données.

Cependant, il ne suffit pas simplement de placer le contrôle sur la feuille. Il faut impérativement renseigner quelques-unes de ses propriétés sans quoi, le contrôle ne marchera pas.

b- Les Propriétés, à modifier, du contrôle Data

Propriété	Description
Connect	Indiquez-y quel type de bases de données vous voulez utiliser: Access, Excel, Paradox, Foxpro...
DataBaseName	Indiquez-y où se trouve votre fichier de base de données sur votre disque.
RecordSource	Indiquez-y le nom de la table à éditer.

Pour pouvoir afficher les données de votre Table, vous aurez sans doute besoin d'un ou plusieurs champs de texte créés à l'aide du contrôle TextBox. Là aussi, vous aurez à modifier quelques-unes de ses propriétés.

c- Les Propriétés, à modifier, du contrôle TextBox

Propriété	Description
DataSource	Indiquez-y le nom du contrôle Data auquel il se rapporte.
DataField	Indiquez-y le nom du champ (de la table) de valeurs auquel il se rapporte.

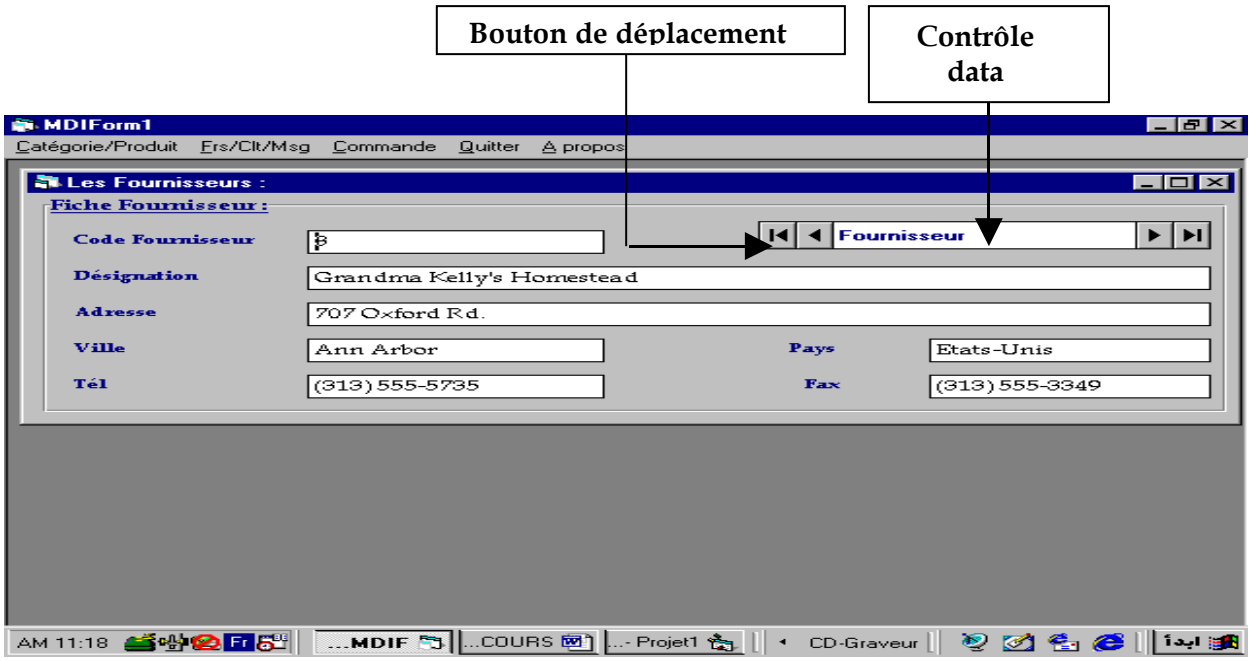
Passons à présent, aux principales méthodes utilisées pour gérer une base de données dans une application.

Méthode	Description
AddNew	Elle sert à créer un nouvel enregistrement. Cependant, Les données nouvellement entrées ne sont pas tout de suite enregistrée à moins que vous n'appuyiez sur une des flèches du contrôle Data. Pour qu'elles soient validées, on pourrait faire appel à la méthode Updaterecord suivi de l'instruction suivante: -- Data1.RecordSet.AddNew -- Data1.recordSet.BookMark=Data1.RecordSet.LastModified qui permet de revenir à la dernière donnée enregistrée ou

	modifiée.
Delete	Elle sert à supprimer un enregistrement donné. -- Data1.RecordSet.Delete
Movefirst, MoveLast, MovePrevious, MoveNext	Elles servent respectivement à revenir au premier enregistrement, à aller au dernier enregistrement, à revenir à l'enregistrement précédent et à aller à l'enregistrement suivant. -- Data1.RecordSet.MoveFirst.
Refresh	Elle sert à rafraîchir la base de donnée. -- Data1.Refresh.
UpdateRecord	Elle sert à mettre à jour une donnée nouvellement enregistrée ou modifiée. -- Data1.UpDateRecord.

Travail à faire :

- Ajouter un contrôle Data à la Feuille ;
- Modifier ces propriétés :
  - Name= « Dt_Fournisseur »
  - Databasename= « c:\Gestion Stock\Bd_Stock.mdb »
  - RecordSource= "Fournisseur"
- Lier les zones de texte au contrôle Dt_Fournisseur ;
  - DataSource= « Dt_Fournisseur » ;
  - Datafield= « Le champ correspondant de la Table » ;
- Exécuter le programme ;
- Utiliser les quatre boutons du contrôle Data pour pouvoir vous déplacer entre les enregistrements :



3- Etape 3 : Ajout du Contrôle DBGrid

Le Contrôle DBGrid permet d'afficher les données d'une table sous forme de tableau (liste). Pour que ce contrôle fonctionne correctement on doit modifier un certain nombre de propriétés.

Propriété	Description
DataSource	Indiquez-y le nom du contrôle Data auquel il se rapporte.
DataField	Lier les noms des champs de la table qui vont figurer dans la liste aux colonnes de la liste

Comment ajouter un nouveau contrôle à la boîte à outils ?

Du menu Projet cliquez sur Composant ;

Dans la boîte de dialogue « Composant », Cochez l'option « **Microsoft Data Bound grid** » ;

Cliquer sur Appliquer ;

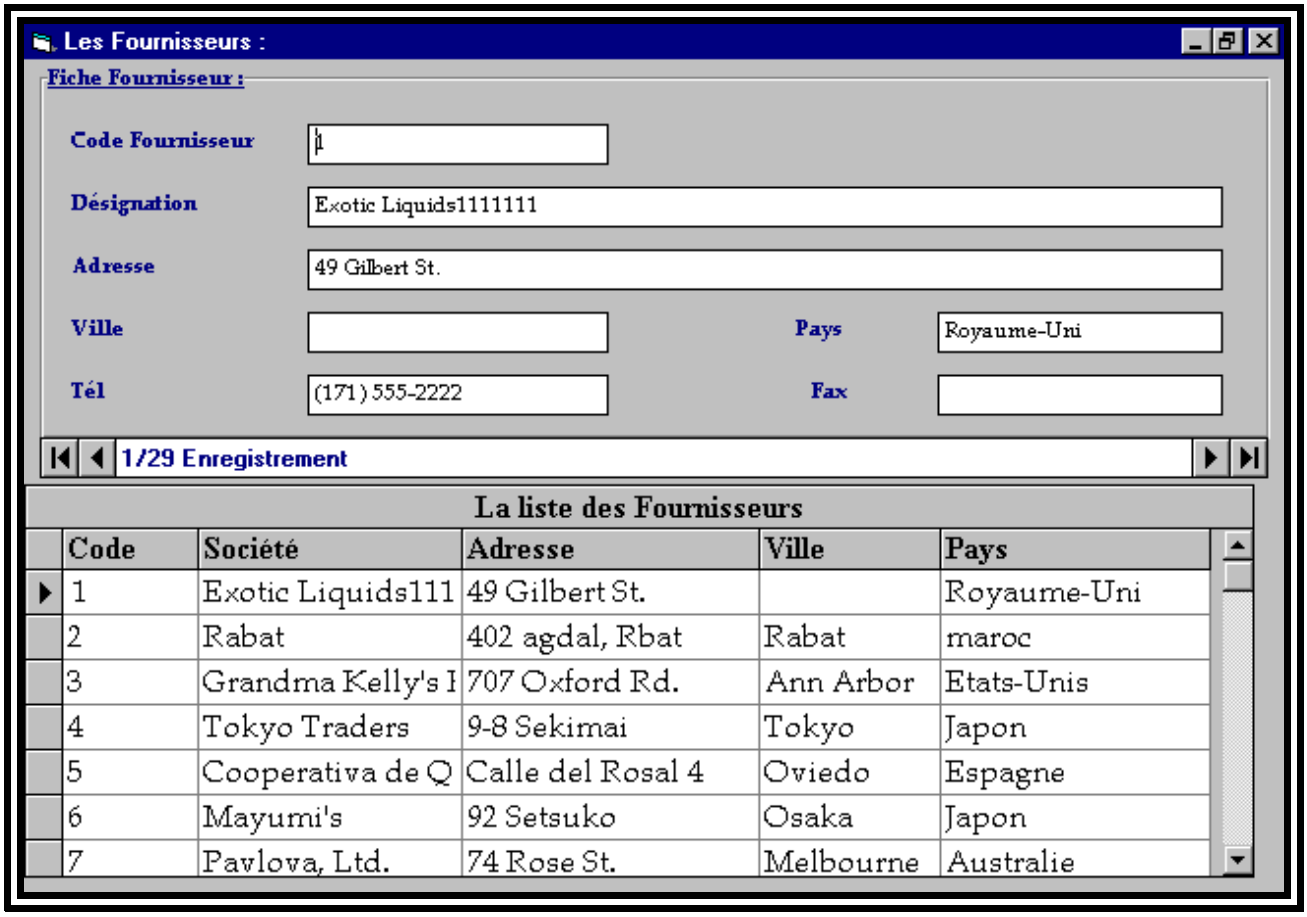
Travail à faire :

- Ajouter le contrôle à la boîte de dialogue ;
- Disposer le contrôle sur la feuille ;
- Modifier les propriétés du contrôle :

- DataSource= « Dt_Fournisseur » ;
- Caption = « La liste des Fournisseurs » ;
- Font = «Police = Book antiqua », «Taille = 12 »
- HeadFont= «Police = times new roman », «Taille = 12 », « attributs = Gras »
- Lier les Colonnes aux champs correspondants ;

➥ Exécuter le programme pour voir le résultat ;

➥ Et voici l'état final de votre feuille ;

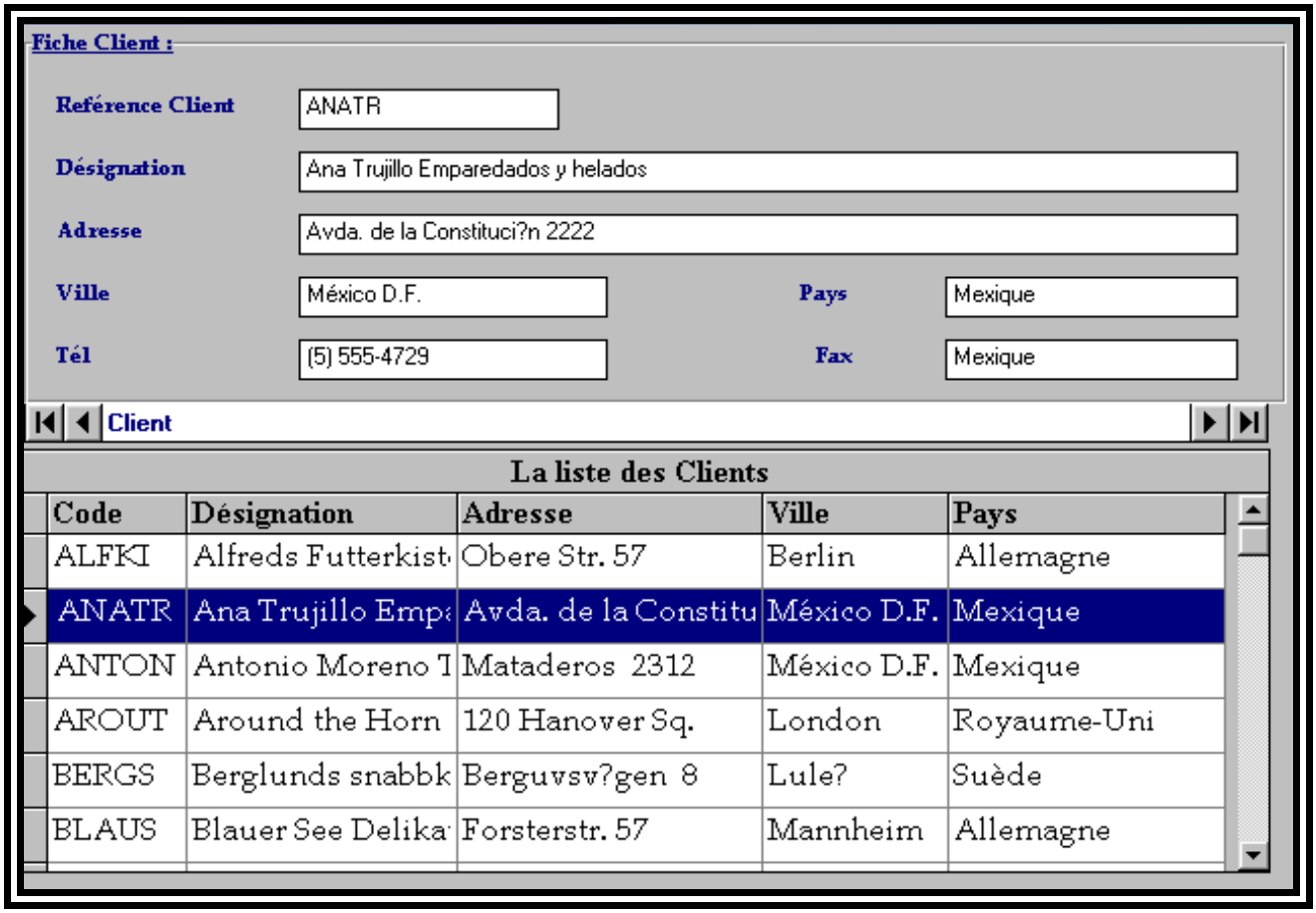


Fin du TP3

EXERCICE 2 : CREATION DE LA FEUILLE CLIENT

La feuille Client sera un écran de consultation/Modification de la table « Client ».

1. Ajouter une nouvelle Form à votre projet ;
2. Nommer cette feuille « Frm_Client » ;
3. Réaliser l'interface de la figure 2;
4. Lier le contrôle Data à la table « Client » ;
5. Enregistrer le projet ;
6. Exécuter l'application ;



Les variables et les constantes

1. Définition des variables

Les variables sont des données ou des valeurs qui peuvent changer à l'intérieur d'une application. C'est pourquoi, il est fort utile de les nommer par un nom, de déclarer quel genre de variables est-ce (nombre entier, nombre décimal, lettres...) et leur affecter, lorsque cela est nécessaire une valeur. La longueur maximale du nom d'une variable est de 255 caractères. Ceux-ci peuvent être des chiffres, des lettres ou autres symboles. Notez que ce nom doit obligatoirement commencer par une lettre.

En effet, Visual basic classe les variables en fonction de la valeur affectée à la variable. Ainsi, une variable déclarée comme du type numérique ne peut pas recevoir une valeur chaîne de caractère, ainsi qu'à l'inverse. Notez que si vous ne déclarez pas une variable, Visual Basic se chargera d'affecter par défaut un type de variable (Variant) à celle-ci. Une variable du type Variant peut aussi bien recevoir des données numériques que des chaînes de caractères. Tout dépend de ce que vous avez affecté à cette variable.

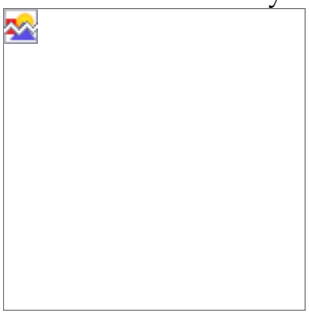
2. Type de variables

A présent, observons de plus près les différents types de variables :

Type de données	Mot clé	Occupe	Limite de valeurs
Octet	Byte	1 octet	0 à 255
Logique	Boolean	2 octets	True(-1) ou False(0)
Entier	Integer	2 octets	-32 768 à 32767
Entier long	Long	4 octets	-2 147 483 648 à 2 147 483 647
Décimal simple	Single	4 octets	Nombre réel avec 7 chiffres après la virgule
Décimal double	Double	8 octets	Nombre réel avec 15 chiffres après la virgule
Monétaire	Currency	8 octets	Nombre réel avec 15 chiffres avant la virgule et 4 chiffres après la virgule
Date	Date	8 octets	1er janvier 100 au 31 décembre 9999
Objet	Object	4 octets	Toute référence à des types Object
Chaîne de caractères	String	10 octets + longueur de chaîne	Chaîne de caractère dont la longueur ne doit pas excéder 2^31 caractères
Variant (avec	Variant	16 octets	toute valeur numérique jusqu'à

chiffres)			l'étendue d'un double
Variant (avec lettres)	Variant	22 octets+longueur de chaîne	Même étendue que pour un String de longueur variable
Défini par l'utilisateur	Type	-	L'étendue de chaque élément est la même que son type de données

Notez que les types de variables les plus utilisés sont : String, Integer, Long, Single, Double et Currency.



3. Déclaration de variables

Pour utiliser des variables, il est normalement obligatoire de les prédéfinir, soit dans la section Déclarations de la liste déroulante Objet, soit en début de procédure ou de fonction. Un programme où les variables sont bien déclarées rend un programme plus facile à comprendre, à lire et surtout à corriger en cas d'erreurs. Certes, il n'est pas obligatoire de les déclarer mais faites-le quand même, c'est un conseil. Si vous êtes prêt à déclarer une variable que vous voulez utilisez mais que vous êtes un étourdi, alors, utilisez simplement l'instruction Option Explicit (à placer dans la section Déclarations de la liste déroulante Objet) qui vous oblige à chaque fois à déclarer toutes vos variables avant de pouvoir exécuter l'application.

3.1 Déclaration explicite d'une variable

Pour déclarer une variable, on utilise l'instruction Dim suivi du nom de la variable puis du type de la variable. Reprenons l'exemple du cours 2 :

```
Dim DateNaissance
Dim Message, Titre As String
```

- Remarquez que la 1ère déclaration ne contient pas d'information sur le type de variable. Si vous déclarez une variable sans donner d'information sur le type de variable que c'est, alors, cette variable (DateNaissance) aura par défaut une variable du type Variant. Si vous avez bien lu l'exemple précédent, vous aurez compris qu'il s'agit ici d'une variable de type Variant (avec chiffres) qui lui est affectée par défaut.
- La 2ème déclaration est par contre explicite. Vous pouvez aussi mettre 2 variables sur une même ligne à condition que le type de variable soit le même pour les 2 et en les séparant par une virgule.

3.2 Déclaration implicite d'une variable

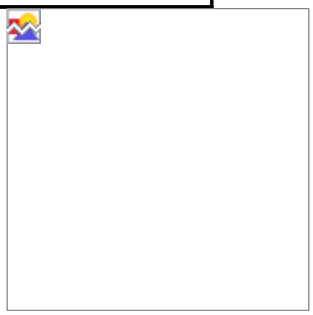
Il existe une autre méthode de déclarer des variables. Pour cela, il suffit d'ajouter juste avant la variable, un symbole spécifique.
Voici la liste des symboles avec le type de variable auxquelles ils se rapportent :

Symbole	Type de variable
%	Integer
&	Long
!	Single
#	Double
@	Currency
$	String

Dans l'exemple ci-après, les deux premières instructions sont équivalentes à la troisième :

```
Dim V1 as Integer
V1 = 25
```

```
V1% = 25
```



4. Portée des variables

En général, toute variable déclarée a une portée limitée. Cette même variable a une valeur nulle au départ. De plus, elle ne s'applique pas forcément à toutes les procédures d'une application. Tout dépend de 2 éléments : la manière de déclarer et l'emplacement de la variable.

- Dans une procédure, si vous déclarez une variable à l'aide de l'instruction Dim, sa portée se trouve limitée seulement à cette procédure. On dit que la variable est locale. Elle est donc initialisée à chaque appel de la procédure et détruite lorsque celle-ci se termine (à moins que vous remplaciez le mot Dim par Static). Elle n'est pas accessible en dehors de la procédure. Vous pouvez remplacer l'instruction Dim par Private, les deux termes étant équivalents s'ils sont placés à l'intérieur d'une procédure.

- Si vous déclarez une variable dans la section Général/Déclarations d'une feuille ou d'un module en utilisant l'instruction Dim (ou Private), la variable est dite locale au module. Cette variable est disponible pour toutes les procédures de la feuille ou du module, mais pas pour les autres feuilles du projet.
- Enfin, si vous déclarez une variable dans la section Général/Déclarations d'un module (et non d'une feuille) en utilisant l'instruction Public au lieu de Dim, elle devient accessible par toutes les feuilles et tous les modules de l'application. On dit qu'elle est globale.

5. Les constantes

Contrairement aux variables dont les valeurs diffèrent souvent, les constantes ont des valeurs fixes. Mais tout comme les variables, on affecte aux constantes, un nom et une valeur qui est fixe. De plus, les constantes se définissent de la même façon que les variables. Tout dépend donc de l'endroit où est défini la constante. Le mot Const peut être précédé de l'option Public pour que toutes les feuilles et modules de l'application puissent y accéder.
Attention cependant à ne pas affecter à une constante des noms identiques aux constantes prédéfinies (VbSystemModal, VbOkOnly, VbArrow...) dans Visual Basic !
Prenons l'exemple du taux de TVA :

```
Const  TVA = 20.6
TotalTTC=PrixHorsTaxe x (1 + (TVA / 100))
```

TP4 : Réalisation d'un Ecran de Mise à jour

Dans ce TP vous n'allez pas créer une nouvelle Feuille mais allez reprendre la Form « Frm_Fournisseur », qui jusqu'à maintenant ne nous a permet que de consulter et modifier les enregistrements de la table « Fournisseur », et vous allez l'améliorer de façon à ce qu'elle soit une Feuille de saisie ou de mise à jours, au niveau de laquelle l'utilisateur pourra, en plus de la consultation et la modification, Ajouter de nouvel enregistrements (fournisseurs) à la table ainsi que supprimer d'autres.

Etape1 : Les événements du contrôle Data :

Le contrôle Data comporte trois événements dont vous pouvez vous servir pour améliorer votre application :

- Validate ;
- Reposition ;

Ces événements vous permettent de modifier le fonctionnement par défaut du contrôle Data.

1- L'événement Validate : Vous avez remarqué dans le TP3 que quand vous modifiez un enregistrement et que vous vous déplacez, à l'aide des boutons de déplacement du contrôle data, vers un autre enregistrement, les modifications effectuées sont valider. Cela se fait grâce à l'événement Validate. Cette événement permet de vérifier les données d'un enregistrement avant de le sauvegarder dans la base de données. Cet événement se produit juste avant que Visual Basic n'écrive dans la base de données les modifications apportées à la table liée et qu'il ne repositionne le pointeur d'enregistrement courant sur un autre enregistrement de la table.

Visual basic vous permet de gérer cet événement. Ainsi vous pouvez Annuler la sauvegarde des modifications ou bien interroger l'utilisateur à chaque fois s'il veut valider ou non les modification effectuée sur la table.

Syntaxe

L'événement Validate utilise la syntaxe suivante :

Private Sub Data1_Validate (*index* **As Integer**, *action* **As Integer**, *save* **As Integer**)

End Sub

L'argument save :

L'argument *save* indique valide l'enregistrement. La valeur de *save* est **True** lorsque les données liées ont été modifiées. Pour annuler la sauvegarde, attribuez la valeur **False ou 0** à *save*.

Travail à faire :

Vous allez Annuler la sauvegarde des modifications l'événement Validate du contrôle Dt_Fournisseur.

- Double cliquez sur le contrôle Dt_Fournisseur ;
- Choisissez l'événement Validate ;
- Affecter la valeur **True** à l'argument save ;

Voici le code :

```
Private Sub Data1_Validate (index As Integer, action As Integer, save As Integer)
    Save = False
End Sub
```

- Exécuter le programme et faites un test ;

2- L'événement Reposition : permet de modifier l'aspect d'une feuille ou d'effectuer toute action nécessaire lorsque vous passez à un nouvel enregistrement.

Cet événement se produit lorsque Visual Basic déplace le pointeur de l'enregistrement courant sur un autre enregistrement de la Table. Il survient également à l'ouverture de la base de données.

Vous pouvez utilisez l'événement Reposition par exemple pour modifier la légende(Propriété Caption)du contrôle Data pour qu'elle affiche le numéro d'enregistrement courant.

Pour afficher le numéro de l'enregistrement courant, utilisez la propriété **AbsolutePosition** de l'objet **Recordset**. Les numéros d'enregistrements sont calculés à partir de zéro, et le numéro du premier enregistrement est donc 0.

L'exemple suivant montre comment afficher le numéro de l'enregistrement courant :

```
Private Sub Data1_Reposition()
    Data1.Caption = " Enregistrement n° "  & Data1.Recordset.AbsolutePosition + 1
End Sub
```

Travail à Faire :

Vous aimeriez bien savoir sur quel enregistrement, de la table, vous êtes placé. Pour cela essayez d'écrire dans l'événement Reposition du contrôle Dt_Fournisseur le code adéquat.

Etape2 : Objet RecordSet et Mise à jours

La Mise à jours d'une table signifie :

- La Modification d'un enregistrement ;
- L'Ajout d'un nouvel enregistrement ;
- La Suppression d'un Enregistrement ;

Dans une application de base de données, le contrôle Data permet de se déplacer dans les enregistrements de la base. Grâce aux boutons du contrôle Data, les utilisateurs peuvent accéder à l'enregistrement précédent ou suivant ou bien passer directement au premier ou au dernier enregistrement.

Cette partie du Cours explique comment le contrôle Data utilise un objet **Recordset** pour afficher les données demandées par l'utilisateur.

Objet Recordset

Avant d'expliquer ce que c'est un Objet recordset vous devez sous familiarisé avec le concept du **jeu d'enregistrements. Un Jeu d'enregistrements** est un ensemble d'enregistrements auquel le contrôle Data fait référence. Le jeu d'enregistrements est stocké en mémoire ou en partie sur disque lorsque cela s'avère nécessaire.

Pour gérer le jeu d'enregistrements, utilisez la propriété **Recordset** du contrôle Data. Le **Recordset** contient l'enregistrement courant. Les informations de ce dernier s'affichent dans les contrôles liés (zones de textes). Pour modifier la position de l'enregistrement courant, vous pouvez cliquer sur le contrôle Data ou écrire du code qui utilise les méthodes de l'objet **Recordset (MoveNext, MoveLast, MovePrevious, MoveFirst)** .

Exemples:

1- Pour se déplacer vers le dernier enregistrement.
Data1.recordset.MoveLast

2- Pour se déplacer vers le premier enregistrement
Data1.recordset.MoveFirst

3- Pour se déplacer vers l'enregistrement. Suivant.
Data1.recordset.MoveNext

4- Permet de se déplacer vers l'enregistrement. Précèdent.
Data1.recordset.MovePrevious

5- Pour Ajouter un nouvel enregistrement
Data1.recordset.AddNew

6- Pour Modifier un enregistrement
Data1.recordset.Edit

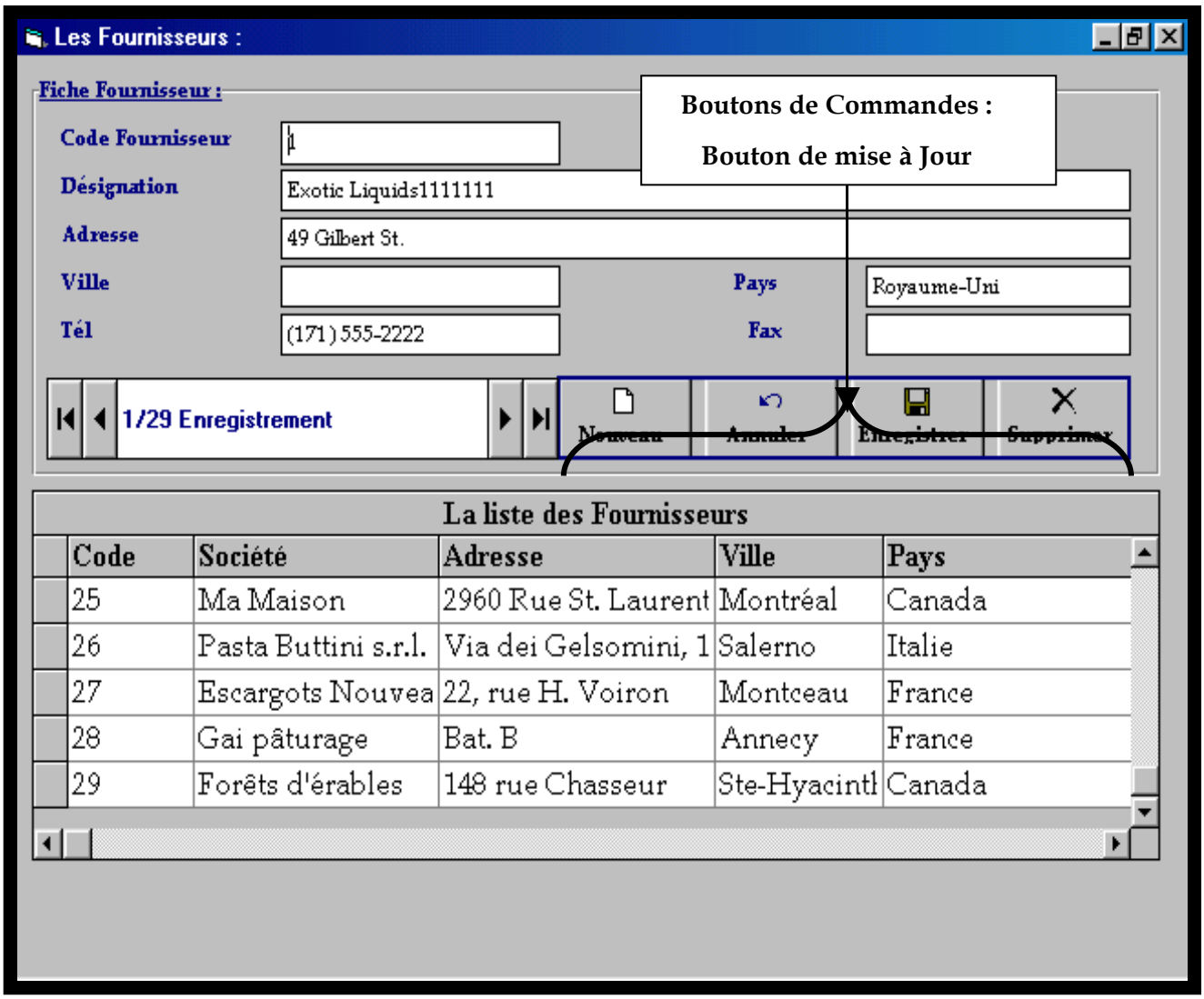
7- Pour enregistrer l'ajout ou la modification d'un enregistrement
Data1.recordset.Update

8- Pour supprimer enregistrement
Data1.Recordset.Delete

9- Pour Annuler l'Ajout ou la Modification d'un enregistrement
Data1.CancelUpdate 'Cette propriété n'est liée qu'à l'objet Data

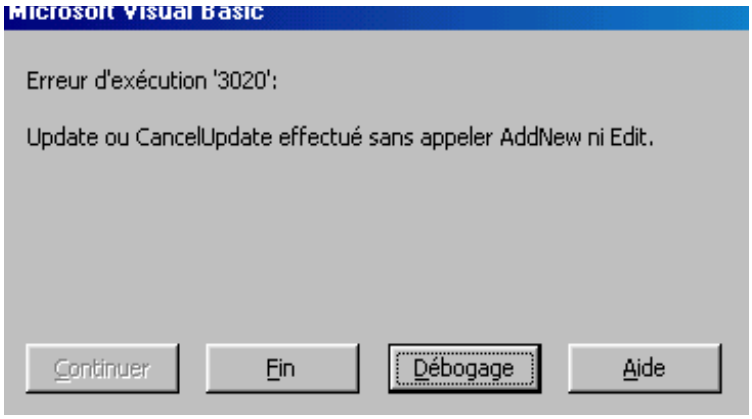
Travail à faire :

- Modifiez l'interface de la Form Frm_Fournisseur de façon à ce qu'elle ait l'aspect suivant :



Vous remarquer qu'on a ajouter à la feuille quatre Boutons.

- Maintenant vous allez ajouter à chaque bouton le code adéquat ;
- Exécuter l'application ;
- Afficher la feuille Fournisseur ;
- Modifier la désignation du fournisseur et cliquer sur le bouton Enregistrer ;
- Un message d'erreur est affiché :



Ce message apparraît quant vous faite appel à la propriété Update ou cancelUpdate càd : Sauvegarder ou annuler la sauvegarde d'un enregistrement sans avoir appeler la propriété AddNew ou Edit

En fait, on ne peut cliquer sur le bouton Enregistrer que si on est en mode modification ou en mode Ajout. On est en mode Ajout qu'on en clique sur le bouton Nouveau (On appel AddNew). Et on est en mode modification quand on appel Edit. Vous remarquer que vous n'avez fait appel à Edit null part dans la feuille Frm_Fournisseur et cela justifie le message que vous avez eu.

Alors ce qu'il faut faire c'est rester toujours en mode modification tant qu'on n'est pas en mode ajout. Pour réaliser ça il faut faire appel à Edit au chargement de la feuille, en cliquant sur le bouton annuler et chaque fois qu'on effectue une opération de Mise à jour afin d'y retourner au mode modification.

- Ajouter la ligne suivante au code de l'événement Activate de l'objet Form ; ***Dt_Fournisseur.Recordset.Edit***
- Ajouter la même ligne au code du Bouton Annuler ;
- Enregistrer et exécuter ;
- Modifier un enregistrement et Enregistrer. Vous remarquer que ça marche ;
- Cliquer sur le bouton annuler et réessayer

Exercice 3

Vous allez appliquer tous ce que vous avez appris dans ce TP4 sur la feuille « Frm_Client »

Les Structure des Tests / La Structure des Boucles

1. Structure des tests

1.1 La structure : If...Then...Else...end If

Voici un exemple de structure simple avec l'instruction If :

```
If condition Then
Instructions 1
else
Instructions 2
End if
```

Interprétation :

Si la condition est vérifiée alors les instructions 1 sont exécutées sinon les instructions 2 sont exécutées à la place. (Notez que l'on peut utiliser des opérateurs logiques And (et) et Or (ou) pour que plusieurs conditions doivent d'abord être vérifiées avant de pouvoir exécuter les instructions suivantes.). Le mot Else et les instructions qui suivent ne sont pas obligatoire.

Voici un autre exemple de structure avec If mais un peu plus complexe:

```
If Condition 1 Then
    Instruction 1
ElseIf Condition 2 Then
    Instruction 2
Else
    Instructions X
End If
```

Interprétation :

Si la condition 1 est vérifiée alors les instructions 1 sont exécutées. Sinon si la condition 2 est vérifiée alors les instructions 2 sont exécutées. Sinon, si aucune de ces deux conditions n'est vérifiée alors les instructions X sont exécutées.

Il existe une autre structure simplifiée de If mais qui moins utilisée:

Variable = **IIf** (Condition, instructions 1, instructions 2)

Interprétation :

Si la condition est vérifiée alors les instructions 1 sont exécutées sinon les instructions 2 sont exécutées à la place.
L'instruction IIf(Immediate If) est utile lorsque qu'il n'y a que une ou deux instructions en fonction d'une condition. Par ailleurs, la valeur retournée par la condition est affectée à une variable.

1.2 La structure Select Case...End Select

Lorsque l'on doit effectuer toute une série de tests, il est préférable d'utiliser la structure Select Case...End Select au lieu de If...Then...End if.
Ainsi, les deux exemples suivants sont équivalents :

```
Select Case Semaine
Case 1
Jour = "Lundi"
Case 2
Jour = "Mardi"
Case 3
Jour = "Mercredi"
Case 4
Jour = "Jeudi"
Case 5
Jour = "Vendredi"
Case 6
Jour = "Samedi"
Case 7
Jour = "Dimanche"
Else
MsgBox "erreur", vbOKOnly , "Note"
End Select
```

```
If  Semaine = 1 Then
    Jour = "Lundi"
ElseIf  Semaine = 2 Then
   Jour = "Mardi"
ElseIf  Semaine = 3 Then
   Jour = "Mercredi"
ElseIf  Semaine = 4 Then
   Jour = "Jeudi"
ElseIf  Semaine = 5 Then
   Jour = "Vendredi"
ElseIf  Semaine = 6 Then
   Jour = "Samedi"
ElseIf  Semaine = 7 Then
   Jour = "Dimanche"
Else
   MsgBox "erreur", vbOKOnly , "Note"
```

```
End If
```

Interprétation :

Si la variable Semaine vaut 1, alors la variable Jour reçoit la valeur Lundi. Si la variable Semaine vaut 1, alors la variable Jour reçoit la valeur Mardi. (Etc.). Si la variable Semaine ne reçoit aucune valeur comprise entre 1 et 7, alors un message indiquant une erreur est affichée.

2. Structure des boucles :

2.1 La structure For...Next :

La structure For...Next est utile lorsqu'on doit répéter plusieurs fois la même instruction.

Exemple :

```
For  J = 1  To  10
T(J) = J
Next
```

Interprétation :

Dans cette boucle, l'instruction "T(J)=J" est répétée 10 fois c'est à dire jusqu'à ce que J soit égale à 10. A chaque passage, l'instruction "T(J)=J" affecte à T(J) les valeurs 1 à 10.

Pour que l'instruction soit exécutée une fois sur deux, vous pouvez utiliser l'instruction Step :

```
For J = 1 To 10 Step 2
T(J) = J
Next
```

Interprétation :

Dans cette boucle, l'instruction "T(J)=J" affecte à T(J) les valeurs 1,3,5,7,9.

2.2 La structure For Each...Next

Cette structure est moins utilisée que la précédente. Elle sert surtout à exécuter des instructions portant sur un tableau.

Exemple :

```
Dim Semaine(1 to 7) As String
```

```
Dim Jour As Variant
Semaine(1) = "Lundi"
'Etc...
For Each Jour In Semaine()
Combo1.AddItem Jour
Next
```

Interprétation :

Pour chaque Jour de la Semaine, on ajoute à la liste combinée, la valeur de la variable Jour jusqu'à ce que l'indice de la semaine soit égal à 7.

2.3 Les structures Do...Loop et While...Wend

Avec ces boucles, le nombre de fois où sont répétées les instructions est indéfini. Les deux exemples qui suivent sont équivalents :

```
i = 1
Do
T(i) = i
i = i + 1
Loop Until i > 10
```

```
i = 1
While i < 10
T(i) = i
i = i + 1
Wend
```

Interprétation :

La variable i est initialisée à 1 au départ. Les instructions qui suivent sont exécutées jusqu'à ce que i soit supérieure à 10. A chaque passage, on affecte à la variable T d'indice i la valeur de i.

Il existe d'autres variantes de ces boucles et qui sont chacune équivalentes :

```
Do
Instructions
Loop While Condition
```

```
Do Until Condition
Instructions
Loop
```

```
Do While Condition
```

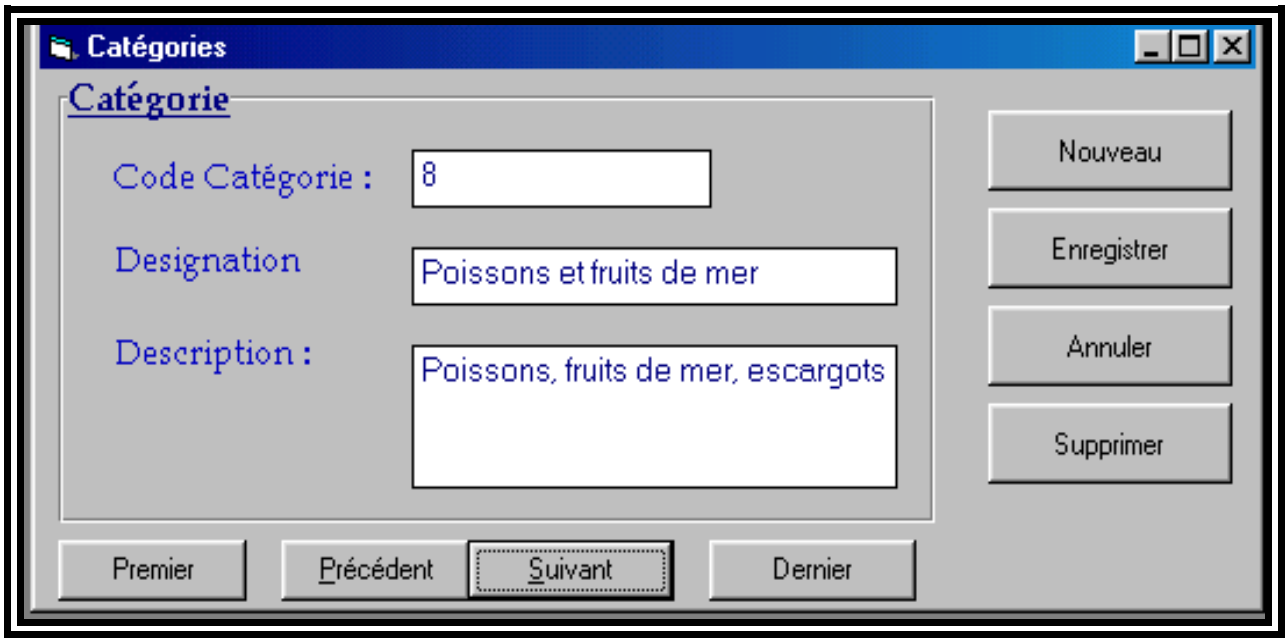
Instructions
Loop

TP5 : Création de la Feuille Catégorie

Dans ce TP vous allez Créer la Feuille Frm_Catégorie et vous allez la lier à la table « Catégorie » en utilisant les objets d'Access au données DAO.

Etape 1: Création de la Form « Frm_Catégorie »

Réaliser l'interface suivante :



Etape 2 : Les Objets DAO

Les objets d'accès aux données (DAO) regroupent un ensemble d'objets qui vous permettent d'accéder aux données de bases de données locales ou distantes et de les manipuler par programme. Vous pouvez utiliser des DAO pour la gestion des bases de données, ainsi que pour celle de leurs objets et de leur structure.

Dans ce TP, vous allez apprendre à utiliser des DAO pour extraire et manipuler les données d'une base de données.

1.1. Ouverture d'une Base de Données

L'ouverture de la base de données constitue la première étape de la création d'une application de base de données. Vous déclarez une variable en tant qu'objet **Database**, puis vous ouvrez la base de données à l'aide de la méthode **OpenDatabase**.

Déclaration d'une base de données

La variable objet **Database** désigne votre base de données. Utilisez l'instruction **Set** pour lui affecter une base de données.

Le code suivant ouvre une base de données

Syntaxe d'ouverture de base de données

```
Dim   MaBase  As Database
Set   MaBase = OpenDatabase (" Chemin de la Base\NomBase.mdb")
```

1.2. Création d'un jeu d'enregistrement (Ouverture de la table)

Après avoir ouvrir une base de données, vous pouvez manipuler les données qu'elle contient en créant un jeu d'enregistrements. Vous déclarez une variable en tant qu'objet **Recordset,** puis vous créez le jeu d'enregistrements à l'aide de la méthode **OpenRecorset**.

Déclaration d'un jeu d'enregistrements

La variable objet **Recordset** désigne votre jeu d'enregistrements. Utilisez l'instruction **Set** pour lui affecter un jeu d'enregistrements.

Le code suivant crée un jeu d'enregistrements.

La Syntaxe de création d'un jeu d'enregistrement :

```
Dim Rs_Table As Recordset
Set Rs_Table = MaBase.OpenRecordset ("Table")
```

1.3. Lier les contrôle d'affichage/Saisie à un jeu d'enregistrement

Les contrôles ne sont pas liés à un jeu d'enregistrements de la même façon qu'ils le sont au contrôle **Data**. Vous devez explicitement copier les données du jeu d'enregistrements dans les contrôles de votre feuille.

L'exemple de code suivant présente une méthode pour copier les données contenues dans un champ dans une zone de texte :

```
ZoneText1.Text = RsTable ("Champ1")
ZoneText2.Text = RsTable ("Champ2")
```

Pour vous déplacer dans un jeu d'enregistrement vous devez utiliser les méthodes de déplacement de l'objet Recordset (MoveFirst, MoveLast etc.)

Comment ajouter une procédure à votre module ?

Travail à faire :

- Ecrivez le code qui permet d'ouvrir la base de données « Bd_Stock » et la table « Catégorie » dans l'événement Load de l'objet Form ;

```
Private Sub Form_Load()
Set db_Stock=OpenDatabase("c:\dalila\vb\gestionstock\Base\BD_Stock. mdb")
Set Rs_Categorie = db_Stock.OpenRecordset ("Catégories")
End Sub
```

- Ecrivez le code suivant pour les boutons de la feuille

  RemplirChamp : pour remplir les zones de texte

  ViderZone : pour vider les zones de texte.

  AffecterZone : pour envoyer le contenu des zones de texte au champs de la table.

```
Private Sub Cmd_Premier_Click()
        If Rs_Categorie.RecordCount = 0 Then
```

```
            MsgBox "La Table est vide", vbInformation
            Exit Sub
          End If
          Rs_Categorie.MoveFirst
          RemplirChamp
  End Sub
```

```
Private Sub Cmd_Dernier_Click()
        If Rs_Categorie.RecordCount = 0 Then
          MsgBox "La Table est vide", vbInformation
          Exit Sub
        End If

        Rs_Categorie.MoveLast
        RemplirChamp
End Sub
```

```
Private Sub Cmd_Precedent_Click()
        If Rs_Categorie.RecordCount = 0 Then
          MsgBox "La Table est vide", vbInformation
          Exit Sub
        End If

        Rs_Categorie.MovePrevious
        If Rs_Categorie.BOF Then
          Beep
          Rs_Categorie.MoveFirst
          MsgBox "C'est le premier enregistrement", vbInformation
        Else
          RemplirChamp
        End If
End Sub
```

```
Private Sub cmd_Suivant_Click()
        If Rs_Categorie.RecordCount = 0 Then
          MsgBox "La Table est vide", vbInformation
          Exit Sub
        End If

        Rs_Categorie.MoveNext
        If Rs_Categorie.EOF Then
          Beep
          Rs_Categorie.MoveLast
          MsgBox "C'est le dernier enregistrement", vbInformation
        Else
          RemplirChamp
        End If
```

```
End Sub
```

```
Private Sub Cmd_Nouveau_Click()
        ViderZone
        Rs_Categorie.AddNew
End Sub
```

```
Private Sub Cmd_Enregistrer_Click()
        If Rs_Categorie.EditMode = dbEditNone Then
          Rs_Categorie.Edit
        End If
        AffecterZone
        Rs_Categorie.Update
End Sub
```

```
Private Sub Cmd_Annuler_Click()
      If Rs_Categorie.RecordCount <> 0 Then
        Rs_Categorie.MoveFirst
        Rs_Categorie.Edit
        RemplirChamp
      End If
End Sub
```

```
Private Sub Cmd_Supprimer_Click()
      Rs_Categorie.Delete
      Rs_Categorie.MoveLast
      If Rs_Categorie.EOF <> Rs_Categorie.BOF Then
        RemplirChamp
      End If
End Sub
```

Erivez le code suivant pour les 3 procédures :

```
Sub RemplirChamp()
        CodCat = Rs_Categorie.Fields("codcat")
        DesCat = Rs_Categorie.Fields("descat")
        DscCat = Rs_Categorie.Fields("Dsccat")
End Sub
```

```
Sub AffecterZone()
        Rs_Categorie.Fields("codcat") = CodCat
        Rs_Categorie.Fields("descat") = DesCat
        Rs_Categorie.Fields("Dsccat") = DscCat
End Sub
```

```
Sub ViderZone()
        CodCat = ""
        DesCat = ""
        DscCat = ""
End Sub
```

Etape 3 : Utiliser un tableur pour l'affichage de données d'un jeu d'enregistrement

1- Le tableur : L'objet MsflexGrid

Tout d'abord, pour pouvoir utiliser le tableur vous devez au préalable placer dans votre boîte d'outils, le contrôles Microsoft Flexgrid Control 6.0. Pour cela, faites un clic droit sur la boîte à outils et choisissez la commande "Composants". Ensuite, cochez la case correspondante au contrôle et validez.

Grâce au contrôle MSFlexGrid et après avoir acquis une certaine expérience, vous pouvez même créer des tableurs semblables à ceux du logiciel Excel de Microsoft. Les propriétés du contrôle MSFlexGrid permettent de manipuler les cellules de ce dernier.

Rows	Elle permet de définir ou de connaître le nombre de lignes du contrôle.
Cols	Elle permet de définir ou de connaître le nombre de colonnes du contrôle.
FixedRows	Elle permet de définir ou de connaître le nombre de lignes fixes du contrôle.
FixedCols	Elle permet de définir ou de connaître le nombre de colonnes fixes du contrôle.
Row	Elle permet de définir ou de connaître les coordonnées de la cellule courante.
Col	Elle permet de définir ou de connaître les coordonnées de la cellule courante.
Text	Elle permet de définir ou de connaître le contenu de la cellule courante.
ColWidth	Elle permet de définir ou de connaître la largeur d'une colonne.
RowHeight	Elle permet de définir ou de connaître la hauteur d'une ligne.
ColAlignment	Elle permet de définir ou de connaître l'alignement des données d'une colonne.
RowSel	Elle permet de sélectionner ou de connaître les coordonnées d'une plage de cellules.
ColSel	Elle permet de sélectionner ou de connaître les coordonnées d'une plage de cellules.

Travail à faire :

Ajouter l'objet Ms FlexGrid à la form « Catégorie »

Modifier ces propriétés :

- Name = « List_Cat »
- Appearance = Flat;
- Cols = 3
- Rows = 2
- FixedRow =1

Ajouter une procédure Entete_Liste à votre module ;

Cette procédure sert à modifier le texte des titres de colonne de la liste ;

Ecrire le code suivant

```
Sub Entete_List()

    '** Commentaire :
    '** Modifier la largeur de chaque colonne
    List_Cat.ColWidth(0) = 1000
    List_Cat.ColWidth(1) = 2000
    List_Cat.ColWidth(2) = 3000

    '** 1 - On sélectionne la première ligne (entête)
    '** 2 - On sélectionne la première colonne
    '** 3 - on écrit ensuite le code dans la méthode "Texte"
    List_Cat.Row = 0
    List_Cat.Col = 0
    List_Cat.Text = " Code "
    List_Cat.Col = 1
    List_Cat.Text = " Désignation "
    List_Cat.Col = 2
    List_Cat.Text = " Description "
End
```

cette procédure sera appelée au chargement de la feuille :

Dans la méthode « Load » de l'objet Form ajouter cette ligne de code

Entete_Liste

Maintenant vous allez ajouter une deuxième procédure « Remplir_List » et vous allez écrire le code permettant d'afficher tous les enregistrements de la table « Catégorie » dans la liste.

```
Sub Remplir_List()
```

```
    Dim NbrEnrg As Integer
    '** Commentaire :
    '** Vous allez commencer par vérifier le nombre d'enregistrement dans
    '   la table "Categorie" Si c'est > 0 alors vous allez les parcourir
    '   tous et les affecter à la liste
    '** Chaque enregistrement occupera une ligne
    If Rs_Categorie.RecordCount = 0 Then
      List_Cat.Clear
      Call Entete_List
      Exit Sub
    End If
    Rs_Categorie.MoveFirst
    NbrEnrg = 1
    Do Until Rs_Categorie.EOF = True
      List_Cat.Rows = NbrEnrg + 1
      List_Cat.Row = NbrEnrg
      List_Cat.Col = 0
      List_Cat.Text = Rs_Categorie("CodCat")
      List_Cat.Col = 1
      List_Cat.Text = Rs_Categorie("DesCat")
      List_Cat.Col = 2
      List_Cat.Text = Rs_Categorie("DscCat")
      NbrEnrg = NbrEnrg + 1
      Rs_Categorie.MoveNext
    Loop
End Sub
```

Exercice 4 :

En se basant sur la structure de la table Produit et sur ce que vous avez appris en travaillant le TP5 :

Réaliser l'interface de la feuille « Frm_Produit ». Cette interface devra être munie de boutons de déplacement, boutons de Mise à Jours, Objet « MsFlexGrid » et devra permettre l'accès aux enregistrements de la table Produit elle devra aussi permettre la mise à jours des données de la table et ceci en utilisant les Objets d'accès au donnés « DAO »

Exécuter le programme et tester votre travail ;

TP6 : La Gestion d'Erreur

Même les applications les mieux conçues ne sont pas à l'abri des erreurs d'exécution. Un utilisateur peut oublier d'insérer une disquette dans un lecteur, un système peut manquer de mémoire ou des fichiers peuvent se trouver ailleurs qu'à l'emplacement prévu. L'ajout de code de gestion d'erreurs efficace à votre application améliorera ses performances.

Vous avez remarqué que chaque fois qu'il y a une erreur dans votre code un message est affiché et l'exécution s'arrête. Vous êtes d'accord que ce n'est pas pratique que chacune erreur surviennent l'application devra s'arrêter. Afin d'éviter ça, il faut créer une routine de gestion d'erreur qui permettra d'annoncer l'erreur sans toutefois bloquer ou arrêter l'exécution.

Alors comment peut-on gérer les erreurs ?

Le processus de gestion d'erreurs comprend les trois étapes suivantes :

1. Activer la récupération d'erreurs en spécifiant l'endroit où doit se poursuivre l'exécution en cas d'erreur.
2. écrivez le code de gestion d'erreurs.
3. Fin de l'exécution du code de gestion d'erreurs.

L'instruction **On Error GoTo** permet d'activer la récupération d'erreurs et de spécifier l'endroit où doit se poursuivre l'exécution en cas d'erreur. Lors d'une erreur d'exécution, l'exécution reprend à partir de l'étiquette spécifiée dans l'instruction **On Error GoTo**. Le gestionnaire d'erreur exécute le code de gestion d'erreurs, qui est suivi d'une instruction **Resume** spécifiant l'endroit où doit reprendre le traitement.

Le code ci-dessous montre comment utiliser l'instruction **On Error Go To**

```
Private Sub save_Click()
   On Error GoTo Gestion_Erreur
   Data1.recordset.update
   Exit Sub

Gestion_Erreur:
   If Err.Number = 3022 Then
      MsgBox "Vous n'êtes ni en mode Ajout ni en mode modification" & chr(13) & _
            " vous ne pouvez pas enregistrer "
      Exit Sub
  Elseif  Err.Number=13 then
         MsgBox  " Vous avez écrit une donnée non correcte. Impossible de la _
               convertir " & chr(13) &  " Veuillez vérifier "
      Exit Sub
  Else
      MsgBox " Une erreur :" & err.Description, vbCritical, " Err.Number"
  End If
End Sub
```

Dans du code de gestion d'erreurs, utilisez les méthodes et propriétés de l'objet **Err** pour identifier une erreur, effacer sa valeur ou déclencher une erreur.

Travail à Faire :

Prenez la Feuille « Frm_Fournisseur » ;

Ajoutez une procédure « Gest_Erreur » ;

Ecrivez le code de l'étiquette Gestion Erreur ;

Dans l'événement clique du bouton Enregistrer, ajouter, au début, l'instruction :

On Error Go To Erreur

Ajouter à la fin l'étiquette Erreur : dans laquelle vous feriez appel à la procédure "Gest_Erreur"

Cette étiquette devra être précédée par Exit sub ;

Lancer l'exécution et tester ;

Exercice :

Ajoutez la routine de Gestion d'erreur à tous les boutons de Mise à Jour ;

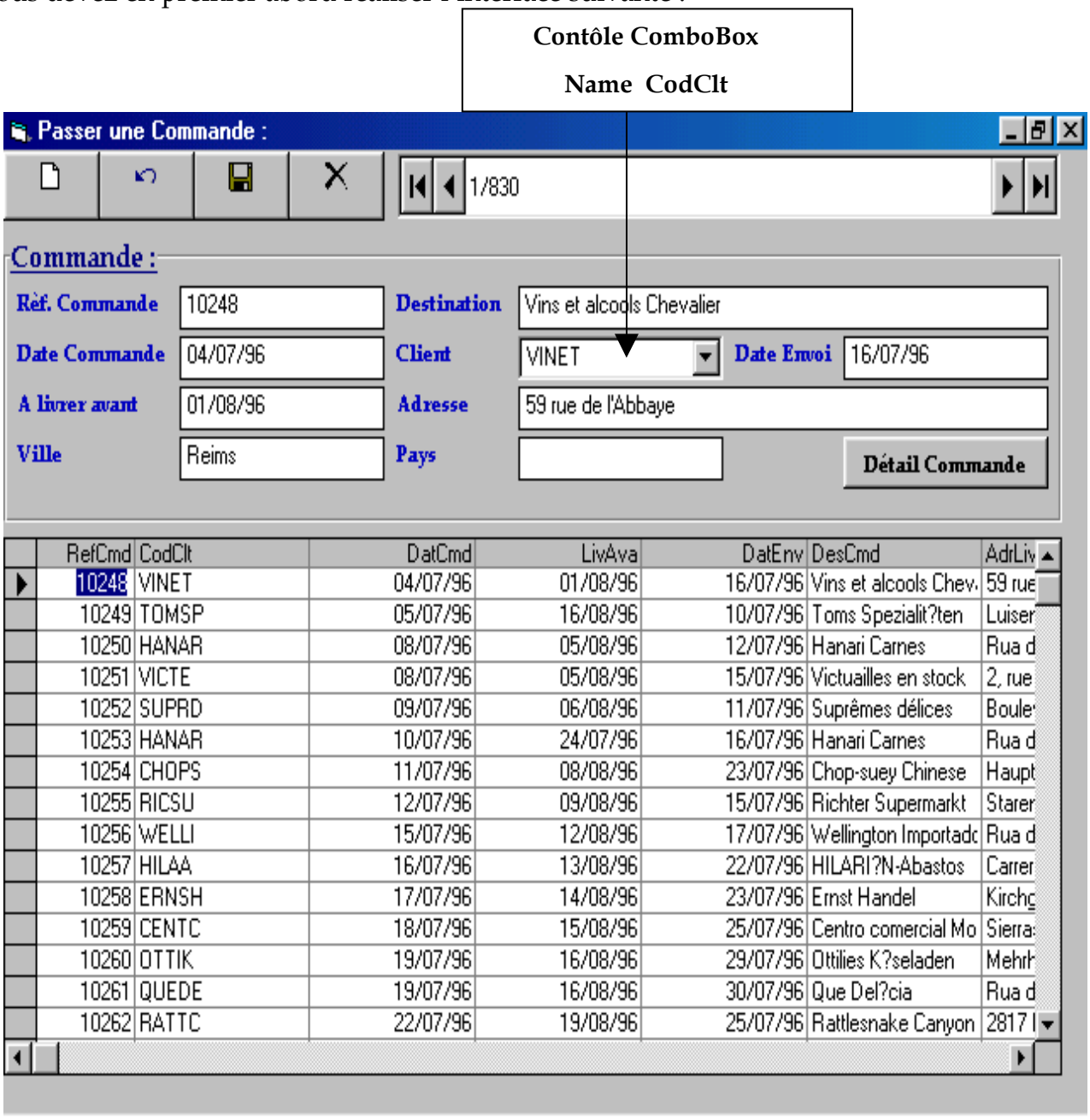
TP7 : Création de la Form Passer_Commande

Quand un client passe une commande, cette commande peut concerner un ou plusieurs produit.

Au niveau de ce TP nous allons réaliser la Feuille Commande qui va gérer toutes les commandes avec leurs détails. Cela signifie quand aura accès à deux Tables minimum(Commande et DétailCommande). Ainsi notre feuille aura une interface un peut spéciale : Présentation feuille Principale /Secondaire.

Etape 1 : Réaliser l'interface

Vous devez en premier abord réaliser l'interface suivante :



- Liez le contrôle Data à la Table Commande ;
- Saisissez le code des boutons de mises à jours ;
- Liez les contrôles DBGrid et TextBox au contrôle data et spécifiez leurs propriétés ;

La liste du Contrôle ComboBox peut être remplit manuellement grâce à la propriété List que vous trouvez dans la fenêtre des propriétés. Elle peut aussi être renseignée à partir d'une table.

Dans le code on pourra ajouter des éléments à la liste grâce à la méthode « AddItem »

La Syntaxe est

ComboBox.AddItem élément

Element : peut être une valeur bien déterminée ou un champs d'objet Recordset ex RecordSet(« field1 »)

Le ComboBox « Codclt » servira à afficher la liste de tous les clients enregistrés dans la table « client » ainsi les données seront fiables et l'utilisateur n'aura qu'à choisir de la liste. Pour remplir la liste du comboBox on va utiliser les objets d'accès aux données DAO.

Le contrôle ComboBox sera aussi lier au contrôle data et jouera le même rôle qu'une zone de texte avec la différence qu'on n'aura pas à saisir mais seulement à choisir un élément de la liste.

- Ajoutez une procédure RemplirList_Clt ;
- Ecrivez le code qui remplira la liste par la liste des références des clients enregistrés dans la table « Client » ;

```
Public Sub RemplirList_Clt()
  '**** Commentaire :
  '*** Déclarer les variables
  Dim BD_Stock As Database
  Dim Rs_Client As Recordset
  '*** Ouvrir la Base de données Stock et la table "Client"
  Set BD_Stock = OpenDatabase("c:\dalila\vb\gestion stock\base\bd_stock.mdb")
  Set Rs_Client = OpenDatabase.OpenRecordset("client")
  '*** On vérifie comme toujours si la table contient des enregistrements
  '    sinon on sort de la procédure
     If Rs_Client.RecordCount = 0 Then Exit Sub
  '*** On cherche dans la table "Client" puis pour chaque
```

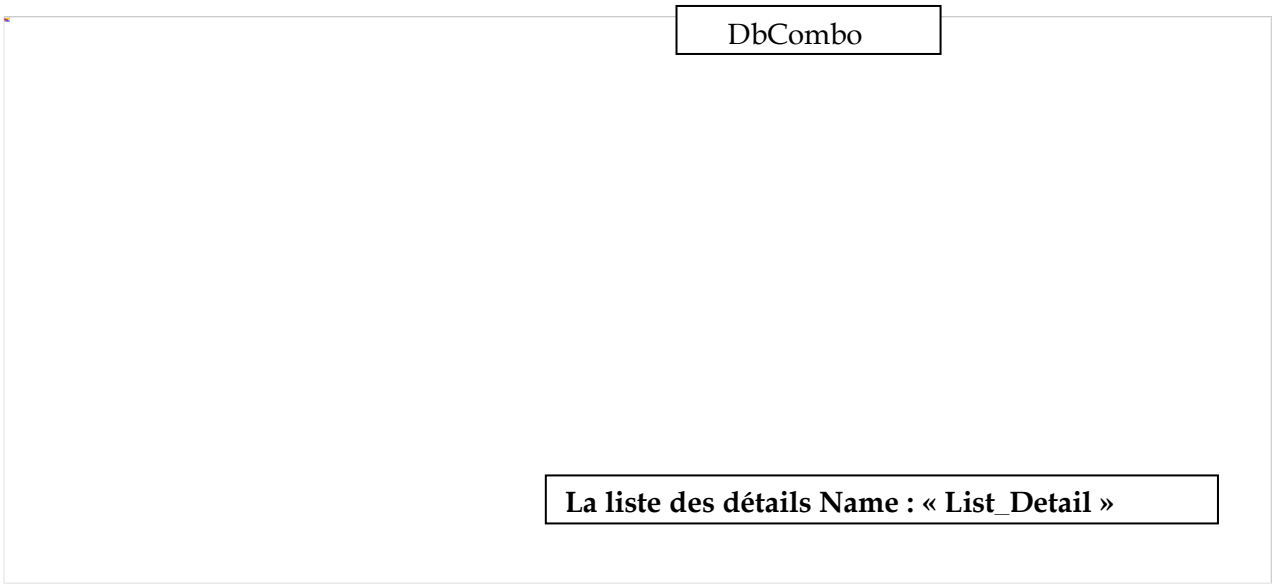
```
'    enregistrement on prend le code client et on
'   l'ajoute à la liste du combobox
Rs_Client.MoveFirst
Do Until Rs_Client.EOF = True
  CodClt.AddItem Rs_Client("codclt")
  Rs_Client.MoveNext
Loop
End Sub
```

- Appeler cette procédure dans l'événement Load de la feuille ;

Etape 2 : Modifier l'interface

Jusqu'à maintenant on n'a fait que créer une simple feuille de saisie et de mise à jours mais ce n'est pas suffisant parce qu'on ne peut pas passer une commande sans spécifier ces détails (càd : les produits concernés et la quantité demandée de chaque produit). Pour cela on va modifier la feuille « Frm_Commande » :

- Ajouter un contrôle Frame et placez le sur le contrôle DBGrid :
  - Name= « Fr_Detail »;
  - Visible = False;
- Placer les contrôles TextBox et commandeBoutonsur le Frame ;
- Ajouter un contrôle DBCombo : Name = « DesPdt » ;



Cet écran sera masqué par défaut il ne sera affiché que si on clique sur le bouton « **DétailCommande** ».

➥ Ecrivez le code qui permet d'afficher la partie détail :

```
Private Sub Cmd_DetCmd_Click()
Fr_Detail.Visible = True
End Sub
```

Une fois l'écran affiché on peut le masquer à tout moment en cliquant sur le bouton « **Fermer** ».

```
Private Sub Bt_Fermer_Click()
Fr_Detail.Visible = False
End Sub
```

Au niveau de cet écran on devrait choisir un produit à partir du DbCombo puis déterminer la quantité commandée. Le total sera calculé automatiquement à partir de la quantité et le prix total.

Une fois on a toutes les informations sur le produit commandé on devrait cliquer sur le bouton Ajouter afin de l'ajouter à la liste des détails de la commande.

Le contrôle DBCombo

Le contrôle **DBCombo** est une liste modifiable dépendante comprenant une zone de liste déroulante qui est automatiquement complétée à partir d'un champ dans un contrôle **Data** attaché, et qui met à jour de façon optionnelle un champ dans une table connexe d'un autre contrôle **Data**

Le contrôle **DBCombo** est différent du contrôle standard **ComboBox**. Alors que la liste du contrôle **ComboBox** est remplie à l'aide de la méthode **AddItem**, le contrôle **DBCombo** est automatiquement complété avec les données provenant d'un champ de l'objet **Recordset** d'un contrôle **Data** auquel il est attaché. Le contrôle standard **ComboBox** doit être complété manuellement à l'aide de la méthode **AddItem**. De plus, le contrôle **DBCombo** a la faculté de mettre à jour un champ à l'intérieur d'un objet **Recordset** connexe qui peut résider dans un contrôle **Data** différent.

Pour qu'un DBCombo Fonctionne il faut modifier un certain nombre de propriétés :

Propriété	Signification
DataSource	Nom du contrôle Data qui est mis à jour lorsqu'une sélection a été faite.
DataField	Nom d'un champ qui est mis à jour dans l'objet Recordset spécifié par la propriété DataSource.
RowSource	Nom du contrôle Data utilisé en tant que source d'éléments

	pour la partie liste du contrôle.
ListField	Nom d'un champ dans l'objet Recordset spécifié par RowSource qui est utilisé pour remplir la liste déroulante. La propriété Listfield n'accepte pas les champs de type LongBinary.
BoundColumn	Nom d'un champ de l'objet Recordset spécifié par RowSource, qui doit être renvoyé à la propriété DataField lorsqu'une sélection a été faite. Le champ BoundColumn n'accepte pas les champs de type LongBinary.
BoundText	Valeur texte du champ BoundColumn. Lorsqu'une sélection a été faite, cette valeur est renvoyée pour mettre à jour l'objet Recordset spécifié par les propriétés DataSource et DataField.

Le DBCombo qu'on a ajouté devra servir à afficher la liste des produits et chaque fois qu'on en choisisse un il affiche sa référence dans la zone de texte CodPdt.

Travail à faire :

- Ajoutez un contrôle data ;
- Modifier les propriétés du contrôle Data :
  - Name =Dt_Produit ;
  - Visible = False;
- Liez le contrôle Data à la table « produits » ;
- Lier le contrôle Dt_Produit et modifier ces propriétés :
  - DataSource = « Dt_Produit » ;
  - RowSource= « Dt_Produit » ;
  - ListField = « NomPdt » ;
  - DataField = « NomPdt » ;
  - BoundColumn = « RefPdt » ;
- Ecrivez le code qui permet d'afficher la référence du produit, dans la zone « RefPdt », chaque fois qu'on sélectionne un produit de la liste du DbCombo;

```
Private Sub NomPdt_Click(Area As Integer)

    '**** Commentaire :

    '***   Cette partie de code permet d'afficher le contenu du champ
    '       déterminé dans la propriété BoundColumn du DBCombo dans une
    '       zone de texte ici "RefPdt"
```

```
    If Area = 2 Then
        Area = 0
        RefPdt = NomPdt.BoundText
    End If
End Sub
```

➥ Executer et tester ;

Vous remarquez que chaque fois que vous sélectionnez un élément de la liste sa référence s'affiche dans le contrôle TextBox « RefPdt »;

Le bouton « DétailCommande » ne sert pas seulement à afficher la partie détail mais aussi à sélectionner de la table « DétailCommande » tous les détails relatifs à la commande sélectionnée dans la partie principale et l'afficher dans la liste des détails.

Afin de ne sélectionner que les détails relatifs à une commande on aura recours aux requêtes SQL.

Le langage de requête structuré SQL

Tout d'abord, il faut que je vous explique ce que c'est que ce langage de requête et à quoi est-ce qu'il sert.

L' SQL("Structured Query Language" ou pour les francophones, "Langage de Requête Structuré") est un langage qui a été créé dans le but de communiquer avec une base de données relationnelle.

Pour ceux qui ne le savent pas, une base de données relationnelle est une base de données où les données sont stockées dans des tables. Ces dernières sont en interaction les unes et les autres.

A présent voyons cela de plus près.

1. Recherche de données

1.1 La syntaxe

L'instruction utilisée pour effectuer une recherche dans une base de données est SELECT. Elle sert à interroger une base et de retourner si elles existent, les données que vous recherchez.

La syntaxe de l'instruction SELECT est la suivante:

SELECT [*|ALL|DISTINCT] [TOP X [PERCENT]] Colonne1,Colonne2
FROM Tables
WHERE (Critère de recherche)|(Critère de jointure)
AND|OR [(Critère de recherche)]
GROUP BY [ALL] Colonne1, Colonne2
HAVING (Critère de recherche)

ORDER BY Colonnes [ASC|DESC]

L'instruction **SELECTE** permet de sélectionner les données à partir d'une ou plusieurs colonnes d'une table.

- L'argument * permet de faire une recherche dans toutes les colonnes de la table donnée.
- L'argument ALL sert à indiquer de retourner toutes les valeurs recherchées même ceux qui sont en double. Par exemple, dans une base de données contenant une liste de tous les clients d'une société, il est parfaitement possible qu'il y ait 2 clients ayant le même nom de famille ou le même prénom. Eh bien, cet argument indique de retourner les données de ces 2 clients. Cet argument est celui par défaut. Il est facultatif.
- L'argument DISTINCT est le contraire de ALL. Il sert à indiquer de ne retourner que les données uniques. Il est facultatif.
- L'argument TOP permet de préciser le nombre d'enregistrements que vous souhaitez recevoir en réponse à votre requête à partir du premier enregistrement. PERCENT indique le pourcentage X d'enregistrements que vous souhaitez retourner. Il est facultatif.
- Les mots Colonne1,Colonne2 Indiquent dans quel(les) colonne(s) de la table ou des tables vous souhaitez effectuez votre recherche. N'oubliez pas la virgule si vous effectuez votre recherche dans plusieurs colonnes.

- La clause FROM sert à indiquer à partir de quel(les) table(s) les données doivent être extraites. Cette clause est indispensable pour toute requête
  - Tables représente le nom des différentes tables dans lesquelles vous désirez faire des recherches.
- La clause WHERE permet d'insérer des critères de recherche dans votre requête. Il est facultatif.
  - Le mot Critère de recherche représente les critères de recherche. Par exemple: "WHERE Client = 'Alphonse'" ou bien "WHERE Prix < '1500'".
  - Le mot Critère de jointure permet d'effectuer une recherche dans une colonne présente dans 2 tables différentes.
- Les opérateurs AND et OR permettent d'insérer un autre critère de recherche. Ils sont facultatifs.
  - Le mot Critère de recherche représente aussi des critères de recherche. Par exemple: "WHERE Client='Alphonse' OR Client='Jean'".
- La clause GROUP BY permet de regrouper des données. Il est facultatif.
  - Pour le mot ALLVoir précédemment.
- La clause HAVING permet d'insérer un critère de recherche pour les données regroupées avec la clause GROUP BY. Il est facultatif.

- Pour le mot Critère de recherche, voir précédemment.

- Enfin, la clause ORDER BY permet de trier les données par colonnes. Il est facultatif.

    - Le mot Colonnes représente le nom des colonnes dans lesquelles sont triées les données.

    - L'argument ASCPermet de trier les données par ordre croissant. C'est l'ordre par défaut. Il est facultatif.

    - L'argument DESC permet de trier les données par ordre décroissant. Il est facultatif.

C'est pas très évident mais avec de la pratique, tout vous paraîtra clair. De plus, vous n'êtes pas obligé d'utiliser tous les clauses et arguments montrés ci-dessus. Voici un exemple simple sur la façon d'utiliser les requêtes avec SQL en s'appuyant sur le contrôle Data qui vous permet de visualiser les données:

```
Data.RecordSource="SELECT * FROM Fichier WHERE Prix<'1500' AND Prix>'3000' ORDER BY Prix"
```

Il vous suffit de placer ce code dans un bouton de commande pour qu'il fasse apparaître la liste de tous les prix compris entre 1500 et 3000 de la table "Fichier". La clause ORDER BY permet de ranger le résultat de votre recherche par ordre croissant(ordre par défaut).

1.2 Les opérateurs

Vous pouvez aussi faire appel à des opérateurs pour spécifier des conditions dans une instruction SQL ou servir de conjonction à plusieurs conditions. Vous vous rappelez, vous en avez déjà vu 2 opérateurs avant: ce sont AND et OR. Ces deux là, sont des opérateurs conjonctifs. Il existe 5 types d'opérateur en tout:

- Les opérateurs de comparaisons servent à vérifier les conditions d'égalité, de non-égalité, les valeurs supérieures à et les valeurs inférieures à. Les voici: =,<>,<,>,<=,>=.

- Les opérateurs conjonctifs permetttent d'effectuer plusieurs conidtions à la fois. Les voici: AND et OR.

- Les opérateurs logiques servent à effectuer des comparaisons. Les voici: IN, NOT IN, BETWEEN, NOT BETWEEN, LIKE, NOT LIKE, IS NULL, IS NOT NULL, EXISTS, NOT EXISTS, ALL et ANY.

- Les opérateurs arithmétiques servent à effectuer des opérations de calcul. Les voici: +,-,* et /.

Le tableau suivant vous montre comment utiliser tous ces opérateurs.

Opérateur	Exemple	Interprétation

=	WHERE Prix='1500'	Retourne les valeurs dont le prix est égal à 1500
<>	WHERE Prix<>'1500'	Retourne les valeurs dont le prix est différent de 1500
<	WHERE Prix<'1500'	Retourne les valeurs dont le prix est strictement inférieur à 1500
>	WHERE Prix>'1500'	Retourne les valeurs dont le prix est strictement supérieur à 1500
<=	WHERE Prix<='1500'	Retourne les valeurs dont le prix est inférieur ou égal à 1500
>=	WHERE Prix>='1500'	Retourne les valeurs dont le prix est supérieur ou égal à 1500
AND	WHERE Prix>'1500' AND Prix<'2000'	Retourne les valeurs dont le prix est compris entre 1500 et 2000 non inclus.
OR	WHERE Prix='1500' OR Prix='2000'	Retourne les valeurs dont le prix est égal à 1500 ou 2000
IN	WHERE Prix IN('1500','2000')	Retourne les valeurs dont le prix est égal à 1500 ou 2000
NOT IN	WHERE Prix NOT IN('1500','2000')	Retourne les valeurs dont le prix est différent de 1500 ou de 2000
BETWEEN	WHERE Prix BETWEEN '1500' AND '2000'	Retourne les valeurs dont le prix est compris entre 1500 et 2000 non inclus.
NOT BETWEEN	WHERE Prix NOT BETWEEN '1500' AND '2000'	Retourne les valeurs dont le prix n'est pas compris entre 1000 et 1500 non inclus.
LIKE	WHERE Nom LIKE 'c*' or LIKE '*a'	Retourne les valeurs dont le nom du client commence par un "c" ou se terminant par un "a"
NOT LIKE	WHERE Nom NOT LIKE '*k*'	Retourne les valeurs dont le nom du client ne contient pas la lettre "k"
IS NULL	WHERE Prix IS NULL	Retourne les valeurs dont aucun prix n'a été fixé
IS NOT NULL	WHERE Prix IS NOT NULL	Retourne les valeurs dont le prix a été fixé
EXISTS	WHERE EXISTS (SELECT Prix FROM Fichier WHERE Prix='1500')	Teste pour vérifier s'il y a des prix égaux à 1500

NOT EXISTS	WHERE NOT EXISTS (SELECT Prix FROM Fichier WHERE Prix='1500')	Teste pour vérifier s'il y a des prix différents à 1500
ALL	WHERE Prix > ALL (SELECT Prix FROM Fichier_concurrent WHERE Lieu='Marseille')	teste pour vérifier s'il y a des produits concurrents se trouvant à Marseille qui coûtent moins cher que le vôtre
ANY	WHERE Prix < ANY (SELECT Prix FROM Fichier_concurrent WHERE Lieu='Marseille')	Teste le prix du produit pour savoir s'il est plus bas que celui du concurrent se trouvant dans à Marseille

1.3 Les fonctions mathématiques

Il est aussi possbile avec le langage SQL d'effectuer des calculs directement à partir des données. Cela se fait à l'aide des fonctions mathématiques suivantes:

Fonction	Utilisation
COUNT()	**Comptabilise le nombre d'enregistrements retourné**
SUM()	**Calcule la somme des valeurs retournées**
AVG()	**Calcule la moyenne des valeurs retournées**
MAX()	**Retourne la plus haute des valeurs trouvées**
MIN()	**Retourne la plus petite des valeurs trouvées**

Prenons quelques exemples pour mieux comprendre:

Plaçons ce bout de code dans un champs de texte.

```
Data.recordsource="SELECT COUNT(Produit) FROM Fichier
```

Ce code permet de voir combien est-ce qu'il y a de produits vendus dans un magasin.
Les autres fonctions fonctionnent de la même manière que cette dernière.

2. Ajout et recherche de données

2.1 Ajout de données

Pour ajouter des données dans une base de données, on utilise l'instruction INSERT.
La syntaxe pour ajouter de nouvelles données avec SQL est la suivante:

```
INSERT INTO Table (Colonnes)
VALUES (valeurs1,valeurs2)
```

- L'instruction INSERT permet d'insérer de nouvelles données.
    - Le mot-clé INTO est spécifique à une base Access. Ne la mettez que si vous utilisez Access.
    - Le mot Table représente la table où doit être inséré ces nouvelles données.
    - Le mot Colonnes représente les colonnes dans lesqquelles sont placées ces nouvelles données.
- La clause VALUES vous permet d'insérer de nouvelles données. C'est ici qu'il faut placer les nouvelles valeurs.
    - Les mots valeurs1, valeurs2 représentent les données qui vont être entrées dans la base.

Par exemple :

```
INSERT INTO Fichier_client(Nom,Prénom,Id) VALUES ('Dupont','Jean','568')
```

Le client Dupont Jean, numéro di'dentification 568, a été ajouté dans la base de données.

2.2 Suppression de données

La suppresssion de données se fait à l'aide de l'instruction DELETE. La syntaxe pour supprimer des données à l'aide de SQl est la suivante :

```
DELETE FROM table WHERE (Critère de recherche)
```

- L'instruction DELETE permet d'effacer des données d'une base.
    - La clause FROM sert à indiquer à partir de quel table les données doivent être détruites.
    - Le mot table représente le nom de la table.
    - La clause WHERE permet d'introduire un critère de recherche.
    - Le mot Critère de recherche permet d'effectuer une recherche afin de déterminer quelles données doivent être effacées de la base.

Par exemple :

```
DELETE FROM Fichier_client WHERE Client='Dupont'
```

Le client Dupont a été supprimé dans la base de données.

3. Mise à jour de données

La mise à jour de vos données se fait à l'aide de l'instruction UPDATE.
La syntaxe est la suivante:

```
UPDATE table SET Colonne='valeur'
WHERE (Critère de recherche)
```

- L'instruction **UPDATE** permet de mettre à jour vos données.
    - Le mot **table** réprésente le nom de la table dans lequel vont être mis à jour des données.
    - La clause **SET** permet d'introduire un critère de recherche.
    - Le mot **Colonne** réprésente le nom de la colonne dans lequel vont être mis à jour des données.
    - Le mot valeur réprésente la nouvelle valeur.
- La clause **WHERE** permet d'introduire un critère de recherche.
    - Le mot Critère de recherche permet d'effectuer une recherche afin de déterminer quelles données doivent être effacées de la base.

Par exemple :

```
UPDATE Table SET date='20/05/01' WHERE Produit='Lait'
```

Tous les enregistrements de la colonne date vont être modifiés par '20/05/01'. Mais avec la clause WHERE, seul les produits laitiers seront concernés.

4. Les transactions

Les transactions permettent de grouper une série d'instruction SQL. Ne croyez pas que ça sert à rien. Cette méthode de regroupement peut s'avérer être très utile lorsque vous manipulez des données sensibles. En effet, lorsque qu'une instruction SQL échoue, vous avez la possilité d'annuler la transaction et de revenir en arrière c'est-à-dire avant que les données soient modifiées.

Cette transaction se fait à l'aide des commandes suivantes :

- BEGINTRANS marque le début d'une transaction.
- COMMITTRANS marque la fin d'une transaction.
- ROLLBACK annule la transaction et vous retrouvez vos données initiales.

Voici la syntaxe d'une transaction :

```
On Error GoTo Erreur
BeginTrans
Série d'instruction
CommitTrans
```

```
Exit Sub
Erreur:
MsgBox Error$
RollBack
Exit Sub
```

Vous avez dû remarquer qu'une routine de traitement d'erreur a été placé dans ce code. C'est dans le cas où la transaction échouerait. Sinon, je pense que vous avez compris son fonctionnement en voyant la syntaxe. Ce n'est pas dur à comprendre.

5. Recherche à partir d'un champ de texte

Voici comment faire pour effectuer une recherche à partir d'un mot tapé dans un champ de texte. Voilà, il faut que vous placiez un champ de texte nommé par exemple, "Text1" et un bouton de commande appelé par exemple, "Chercher". Placez le bout de code suivant dans la procédure du bouton:

```
valeur = "SELECT * FROM Liste WHERE Produit LIKE " + " ' " + Text1.Text + "
Data.RecordSource = valeur
Data.refresh
```

Travail à faire :

Ajoutez une procédure « RemplirList_detail » qui permet de selectionner de la table « detailCommande » les détails relatifs à la commande et de l'afficher dans l'objet MsFlexGrid.

```
Public Sub RemplirList_Detail()
    Dim NbrEnrg As Integer
    Dim Requet As String
    Dim Bd_Stock As Database
    Dim Rs_Detail As Recordset

    '**** Ouvrir la Base Stock
    '** Remarque : On ne peut pas faire une requete sur une base de données
    '   sans ouvrir cette dernière
    Set Bd_Stock = OpenDatabase("c:\dalila\vb\gestion stock\base\bd_stock.mdb")

    '** Executer la requet de selection
    '** Explication : on selectionne les enregistrements de la table
    '                 "detailCommande" dont le champ refCmd est égal
    '                 au critère (le critère c'est le contenu de la zone de
    '                 texte "RefCmd", et on les met dans la chaîne Requet
    Requet = "select * from detailcommande where refcmd= " & RefCmd.Text & " "
```

```
'** On ouvre requet en tant qu'objet recordset "Rs_detail"
Set Rs_Detail = Bd_Stock.OpenRecordset(Requet)

'** Vous allez commencer par verifier le nombre d'enregistrement dans
'   le recordset "Rs_Detail" Si c'est > 0 alors vous allez les parcourirs
'   tous et les affecter à la liste
'** Chaque enregistrement occuppera une ligne
If Rs_Detail.RecordCount = 0 Then
  List_Detail.Clear
  Call Entete_List
  Exit Sub
End If
Rs_Detail.MoveFirst
NbrEnrg = 1
Do Until Rs_Detail.EOF = True
  List_Detail.Rows = NbrEnrg + 1
  List_Detail.Row = NbrEnrg
  List_Detail.Col = 0
  List_Detail.Text = Rs_Detail("refpdt")
  List_Detail.Col = 1
  List_Detail.Text = Rs_Detail("qtecmdpdt")
  List_Detail.Col = 2
  List_Detail.Text = Rs_Detail("PUPdt")
  NbrEnrg = NbrEnrg + 1
  Rs_Detail.MoveNext
Loop
Rs_Detail.MoveFirst
```

End Sub

- Appeler cette procédure dans le code de l'événement clique du Bouton « détail Commande ».
- Appeler cette procedure dans l'évenement reposition du contrôle Dt_Commande ainsi même si on se déplace d'une commande à une autre la liste sera mise à jour automatiquement.
- Executer et tester.

Besoin :

Si on clique sur une ligne de la liste détails « le contenu devra être afficher dans les zones de textes ainsi on pourra le modifier (ex : modifier la quantité commandée) le but est de pouvoir modifier et envoyer les modifications à la liste.

- A l'évenement Clique de la liste vous allez ajouter le code suivant qui permet de detecter la ligne choisie, de la liste, par l'utilisateur et afficher les données dans les zones de textes.

```
Private Sub List_Detail_Click()

'*** stocker dans la variable Pos le numéro de la ligne sur laquelle
'   l'utilisateur a cliqué
```

```
    Pos = List_Detail.Row
    '*** Si pos=0 cela signifie que l'utilisateur  a cliqué sur la ligne
    '    d'entête
    If Pos = 0 Then Exit Sub
    '*** Si pos <>0 alors on affecte les données de la ligne pos aux zones de
    '    texte
    List_Detail.Row = Pos
    List_Detail.Col = 0
    RefPdt.Text = List_Detail.Text
    List_Detail.Col = 1
    QteCmdPdt.Text = List_Detail.Text
    List_Detail.Col = 2
    PUPdt.Text = List_Detail.Text
End Sub
```

Si vous tester maintenant vous remarquez que ça marche ;

➥ Une fois on a modifier, on voudrait mettre à jours la ligne dans la liste, pour cela ajouter le code suivant dans le bouton ajouter.

```
Private Sub Bt_Ajouter_Click()

    '*** si la ligne selectionnée est égale à 0 alors on quitte la procédure
    If Pos = 0 Then Exit Sub
    '*** Si une ligne a été selectionnée de la liste la valeur de pos est
    '    comprise entre 1 est le nombre total des lignes et alors on peut
    '    modifier
    If 1 <= Pos <= List_Detail.Rows Then
    '-- on se positionne sur la ligne Pos
       List_Detail.Row = Pos
       List_Detail.Col = 0
       List_Detail.Text = RefPdt.Text
       List_Detail.Col = 1
       List_Detail.Text = QteCmdPdt.Text
       List_Detail.Col = 2
       List_Detail.Text = PUPdt.Text
    '*** une fois les modifications sont effectuées, les zones doivent être vider
    '    de textes et modifier la valeur de pos
       Vider_detail
       Pos = List_Detail.Rows + 1
    End If
End Sub
```

Ce code nous permettra aussi d'ajouter une nouvelle ligne à la liste

➥ On devrait aussi pouvoir enlever (càd : supprimer) une ligne de la liste. Ecrivez le code qui permet la suppression d'une ligne de la liste détail dans l'événement Clique du bouton Enlever ;

```
Private Sub Bt_Enlever_Click()
```

```
    If Pos = 0 Then Exit Sub
    List_Detail.RemoveItem
End Sub
```

- Executer et tester ;
- Ce qu'on voudrait maintenant c'est pouvoir enregistrer une nouvelle commande avec ses détails ainsi que les modifications effectuées sur une commandes existante. Puisque vous avez déjà écrit le code d'enregistrement d'une commande dans le bouton « Cmd_Enregistrer », vous allez l'améliorer.

```
Private Sub Cmd_Enregistrer_Click()

  Dim db_Stock As Database
  Dim Rs_Detail As Recordset
  '*** Enregistrer l'ajout ou la modification d'une commande dans la table commande
  Dt_Cmd.Recordset.Update

  '*** Ouvrir la Base de données Stock et la table "Détails Commandes"
  Set db_Stock = OpenDatabase("c:\dalila\vb\gestion stock\base\bd_stock.mdb")
  Set Rs_Detail = db_Stock.OpenRecordset("Détails Commandes")

  '*** Si on est en mode modification d'une commande alors on devrait supprimer ses
  '     détails puis les réenregistrer par la suite
  If Dt_Cmd.Recordset.EditMode = dbEditprogress Then
    If Fr_Detail.Visible = True Then
      '*** On va executer une requet d'action qui supprimera de la table
      '    détail tous les détails de la commande en cours
      db_Stock.Execute "delete * [détails commandes] where " & _
                  "refcmd='" & RefCmd.Text & "'", dbFailOnError
    End If
  End If
  '*** qu'on soit en mode ajout ou modification
  '   le contenu de la liste détail est ajouté à
  '   la table détail commande

  If Dt_Cmd.Recordset.EditMode = dbEditAdd Then
    If Fr_Detail.Visible = True Then
      List_Detail.Row = 1
      Do Until List_Detail.Row = List_Detail.Rows
        Rs_Detail.AddNew
        List_Detail.Col = 0
        Rs_Detail("refpdt") = List_Detail.Text
        List_Detail.Col = 1
        Rs_Detail("qtecmdpdt") = List_Detail.Text
        List_Detail.Col = 2
        Rs_Detail("pupdt") = List_Detail.Text
        Rs_Detail("refcmd") = RefCmd.Text
        Rs_Detail.Update
        List_Detail.Row = List_Detail.Row + 1
      Loop
```

```
        End If
    End If
End Sub
```

C'est le code qui permettra de pendre les données de la liste et les enregistrer dans la table « Détail Commandes ».

- Quand on supprime une commande on devrait aussi supprimer ses détails pour cela vous allez améliorer le code du bouton «supprimer » en y ajoutant une requet action permettant la suppression dés détails d'une commande de la table « détails commandes ».