VBA pour Access 2007 & 2010

Guide de formation avec cas pratiques

Daniel-Jean David

Tsoft EDITEUR

EYROLLES

Avant-propos

Conçu par des formateurs expérimentés, cet ouvrage vous permettra d'acquérir de bonnes bases pour développer avec Microsoft VBA pour Access. Il s'adresse à des utilisateurs avancés de Microsoft Access qui veulent créer des applications utilisant les outils et les objets Access.

Les versions successives de Microsoft Access 2000, 2002, 2003, 2007 puis 2010 ont surtout apporté des changements aux commandes directes d'Access. Le langage VBA n'a connu que peu d'évolution au niveau de sa syntaxe, et les rares changements apportés au modèle d'objet Access ne concernent que des éléments marginaux que nous n'abordons pas dans ce livre.

Dans Access 2010 ou 2007, les fichiers portent l'extension `accdb` ; dans les versions antérieures, ils portent l'extension `mdb`. À la lecture de ce livre remplacez `accdb` par `mdb` si vous utilisez une version antérieure à 2007.

Fiches pratiques Ce manuel commence par présenter sous forme de fiches pratiques les « briques de base » de la programmation avec Microsoft VBA pour Access. Ces fiches pratiques peuvent être utilisées soit dans une démarche d'apprentissage pas à pas, soit au fur et à mesure de vos besoins, lors de la réalisation de vos applications avec Access VBA.

Méthodologie Une deuxième partie fournit des bases méthodologiques et des exemples réutilisables dans vos programmes. Tous les exemples donnés sont « passe-partout », indépendants de toute version. Nous insistons plutôt sur les aspects « stratégie de la programmation » qui ne doivent pas reposer sur des détails de langage.

Cas pratiques La troisième partie vous propose des cas pratiques à réaliser par vous-même pour acquérir un savoir-faire en programmation VBA pour Access. Cette partie vous aidera à développer des applications en mettant en œuvre les techniques et méthodes étudiées dans les parties précédentes. Tous les cas traités sont « passe-partout », indépendants de toute version.

- Ces cas pratiques constituent autant d'étapes d'un parcours de formation ; la réalisation de ce parcours permet de s'initier seul en autoformation.
- Un formateur pourra aussi utiliser ces cas pratiques pour animer une formation à la programmation VBA pour Access. Mis à la disposition des apprenants, ce parcours permet à chaque élève de progresser à sa vitesse et de poser ses questions au formateur sans ralentir la cadence des autres élèves.

Aide-mémoire La quatrième partie constitue un aide-mémoire utile lorsque vous écrivez du code VBA pour Access, pour retrouver des informations qu'on ne connaît pas par cœur : liste des mots-clés, désignation des touches, principales propriétés…

Vous pouvez télécharger des exemples de code et de données qui ont servi à illustrer cet ouvrage ainsi que les données pour les cas pratiques depuis le site www.editions-eyrolles.com. Rendez-vous sur la page de cet ouvrage, référence **G12992**, et sélectionnez les fichiers à télécharger.

Conventions typographiques

Actions à effectuer

Les commandes de menus sont en italique, séparées par des tirets : *Fichier – Ouvrir*.

Les commandes du ruban sont sous la forme 🗀*Onglet - [Groupe] - Commande.* Il est possible d'ouvrir la boîte de dialogue du groupe en cliquant sur le déclencheur de dialogue ⇲, s'il existe.

Une suite d'actions à effectuer est présentée avec des puces :

- *Affichage* (signifie cliquez sur le menu *Affichage*)
- Cliquez sur la fenêtre à afficher

Une énumération ou une alternative est présentée avec des tirets : Par exemple :

- soit par un nombre
- soit par <nombre1> To <nombre 2>

L'action de frappe de touche est représentée par la touche ainsi : F11.

L'action de frappe d'une combinaison de touches est représentée ainsi : Alt+F11.

L'action de cliquer sur un bouton est représentée ainsi : Ok. **Fichier** représente l'onglet (à gauche de l'onglet Accueil) qui remplace le bouton Office de la version 2007.

Les onglets sont entre guillemets : « Général » ou on précise : Onglet *Général*.

Les cases à cocher sont marquées ainsi : ☑ (il faut la cocher), ☐ (il faut la décocher).

Les boutons radio sont marqués ainsi : ⊙ (choisi), ○ (non choisi).

Extraits de programme

Les extraits de programme sont représentés comme suit :

```
Sub exemple()
Dim x As Integer
   x=3
End Sub
```

Le trait figure la marge. Les indentations (décalages comme pour `x=3`) doivent être respectées.

Dans les descriptions de syntaxe

Une désignation générique d'un élément est présentée entre <> ; dans une instruction véritable, elle doit être remplacée par un élément de syntaxe correcte jouant ce rôle ; une définition générique sera le plus souvent suivie d'un exemple réel en caractères `Courier`. Par exemple, La déclaration d'une variable est de la forme : `Dim <variable> As <type>` Ex. : `Dim x as Integer`

Dans une description, un élément facultatif est présenté entre [ ] (qui ne doivent pas être tapés) :
`For <variable>=<début> To <fin> [Step <pas>]`

Une répétition facultative est présentée comme suit :

`Dim <variable> As <type>[,<variable> As <type> [,…]]`

La place des virgules et des crochets montre que chaque élément facultatif, en plus du premier, doit être précédé de la virgule qui le sépare du précédent. Les [ ] les plus internes peuvent être absents.

Abréviations

BD :	Base de données	VB :	Visual Basic sans application hôte
BDi :	Boîte de dialogue/Formulaire	VBA :	Visual Basic Applications
désign. :	désignation	VBAA :	Visual Basic Applications Access

Table des matières

PARTIE 1
APPRENTISSAGE

Création d'un Programme

1

Les macros

Conversion de macro en VBA

Écriture des instructions VBA : l'éditeur VBA

Règles fondamentales de présentation

Projets, différentes sortes de modules

Options de projets

Les différentes sortes d'instructions

Les menus de l'éditeur VBA

Les macros offrent le premier moyen de créer des séquences de commandes afin de pouvoir les répéter à volonté sans avoir à les retaper. Pour son déclenchement, une telle séquence est le plus souvent associée à un événement qui arrive à propos d'un contrôle dans un formulaire.

On peut même créer une application complète avec un ensemble de macros, encore que, pour cela VBA offre plus de possibilités.

Attention ! Note valable pour toute la partie programmation. Les macros et les programmes VBA ne peuvent fonctionner que si, à l'ouverture de la base de données, en réponse à « Avertissement de sécurité » (sous le ruban), vous cliquez sur Activer le contenu.

CRÉER UNE MACRO

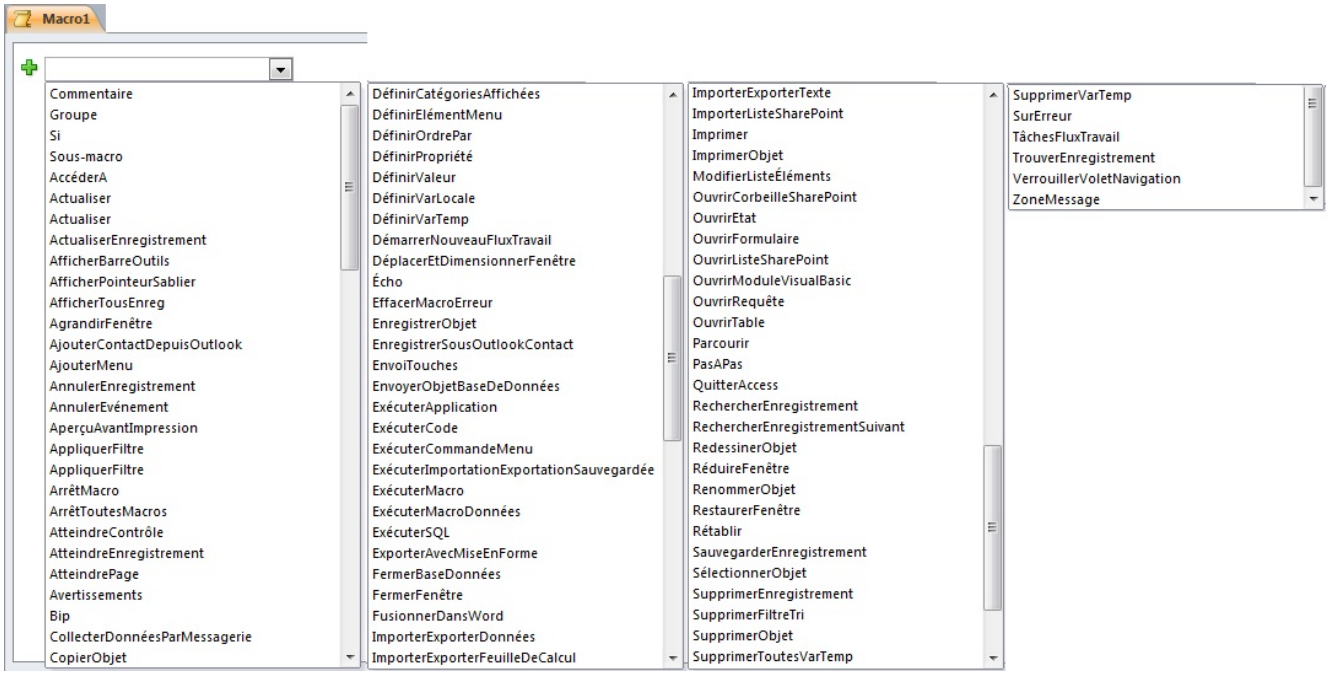
Faites une copie de la base que vous avez en téléchargement CarnetdAdresses.accdb ; nous suggérons le nom CarnetdAdresses_mac.accdb : c'est en tout cas le nom du fichier exemple final qui est à votre disposition en téléchargement.

Lorsque vous voulez utiliser le formulaire Saisie des Amis pour entrer un nouvel enregistrement, le problème est que l'on n'est pas d'emblée positionné sur l'enregistrement vide destiné à recevoir les nouvelles données. Au lieu d'ouvrir le formulaire, vous allez appeler une macro qui effectue la séquence d'opérations suivante :

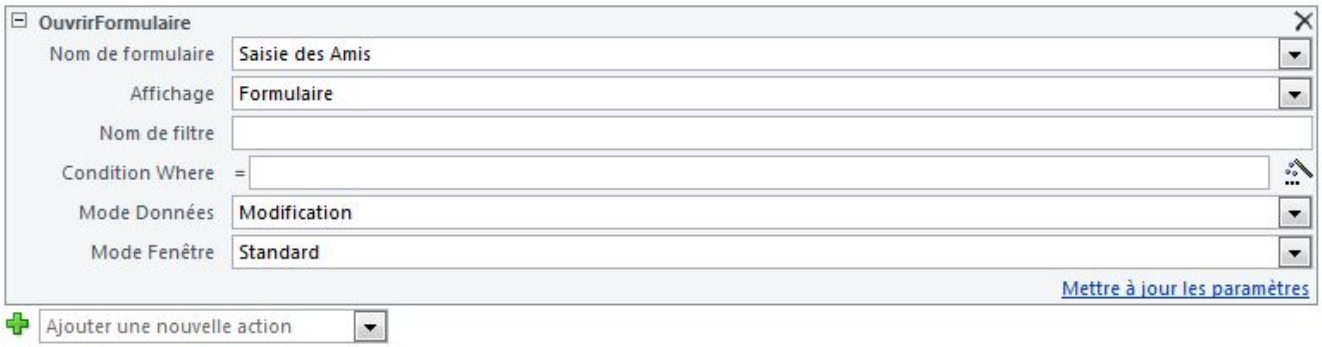
- ouvrir le formulaire

- aller sur le nouvel enregistrement à créer

1 - Appelez 🗀*Créer-[Macros et code]-Macro*. Il vient une zone d'entrée avec liste déroulante des actions possibles. Il apparaît aussi un volet *Catalogue* d'actions que nous n'utilisons pas pour le moment et que vous pouvez masquer. Cliquez sur la flèche descendante pour faire dérouler la liste.



2 - Choisissez OuvrirFormulaire. Vous pouvez alors définir des arguments de l'action dans la grille qui apparaît. Dans Nom de formulaire, une liste déroulante vous donne à choisir entre les formulaires existants : choisissez Saisie des Amis. Pour Mode Données, adoptez Modification et gardez Standard pour Mode Fenêtre.



3 - Cliquez sur *Mettre à jour les paramètres* puis sur le bouton [–] en haut à gauche. La grille se réduit.

4 - On a un nouvel exemplaire de la liste déroulante pour implanter une commande. L'exécution de la macro consiste à effectuer ces commandes à la suite les unes des autres. Comme deuxième action, choisissez AtteindreEnregistrement. Comme valeur du paramètre Enregistrement, spécifiez Nouveau.

5 - Cliquez du bouton droit sur l'onglet Macro1 et choisissez Enregistrer. Enregistrez sous le nom Ajouter Ami.

La macro que nous venons d'écrire est très simple. Le langage des macros permet des traitements relativement élaborés. On voit sur la figure que les actions possibles appartiennent à différentes catégories.

Actions sur un objet

Ouvrir ou fermer une table, un formulaire, un état, une requête.

Agir sur la fenêtre d'un objet : agrandir, réduire, dimensionner, déplacer, sélectionner, redessiner etc.

Gérer les données et les enregistrements

AtteindreEnregistrement, TrouverEnregistrement, TrouverSuivant etc.

Divers

Bip, BoîteMsg, Exécuter : Code (fonction VBA), Commande (toute commande de menu Access), Macro (appelle une autre macro), ArrêtMacro et même Quitter.

Par ailleurs, une action peut être assujettie à une condition (ligne Condition Where), donc effectuée ou non et, même, on peut selon une condition effectuer une action ou une autre.

DÉCLENCHEMENT D'UNE MACRO

Il y a six manières de démarrer une macro.

1) L'appeler depuis une autre macro par l'action ExécuterMacro. Bien sûr la macro appelante doit être déclenchée par une autre méthode.

2) Double-clic sur son nom dans le volet navigation.

3) Lorsqu'elle est en mode création, *Outils de macro Créer-[Outils]-Exécuter (!)*.

4) On peut ajouter un bouton au ruban et l'associer au déclenchement de la macro. C'est un cas particulier de la méthode 5.

5) On peut associer le déclenchement à un événement lié à un contrôle ou un formulaire, comme clic, changement de valeur, arrivée de la sélection etc.

6) Si la macro a le nom AutoExec, elle démarre automatiquement à l'ouverture de la base.

Les événements mis en jeu sont très nombreux. Ils concernent :

- Formulaire ou état

  Ouverture, fermeture, activation.

- Enregistrement

  Insertion, mise à jour, suppression.

- Contrôles

  Entrée, sortie, absence de donnée dans une liste modifiable, appui de touche et, le plus utilisé, clic sur un bouton.

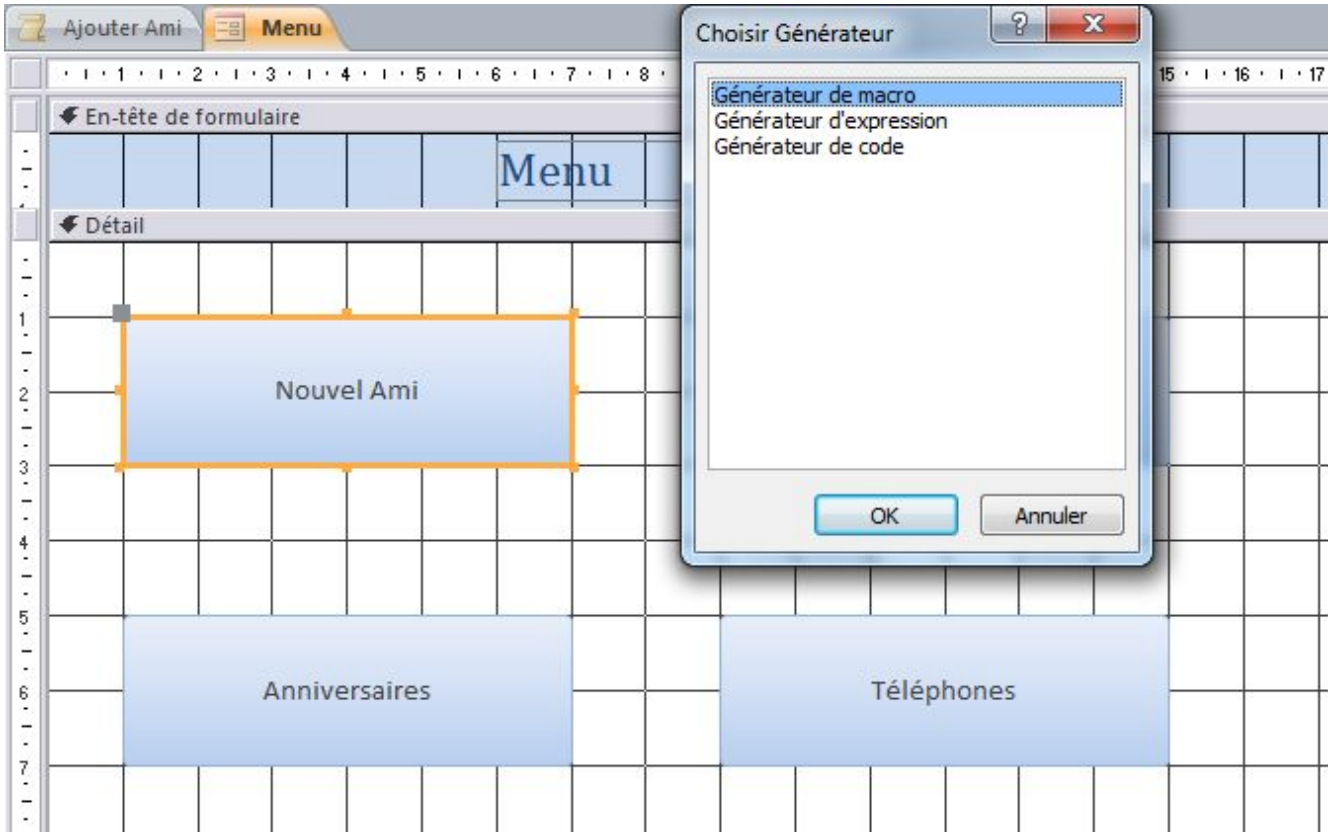
UNE APPLICATION AVEC MACROS

Vous allez transformer votre base *CarnetdAdresses_mac* en application complète de gestion de vos amis. Commencez par créer un formulaire qui servira de menu.

1 - Appelez 🗀*Créer-[Formulaires]-Création de Formulaire*. Installez le titre Menu.

2 - Installez quatre boutons avec les libellés Nouvel Ami, Affichage, Anniversaires et Téléphones. Sauvegardez le formulaire sous le nom Menu. Pour chacun, cliquez sur Annuler pour quitter l'Assistant qui apparaît.

3 - Cliquez sur Nouvel Ami. Cliquez du bouton droit sur le carré marron et choisissez Créer code événement. Choisissez Générateur de macro.



4 - On arrive à l'écran de création de macro. Spécifiez les actions ExécuterMacro Nom=Ajouter Ami puis FermerFenêtre Type d'objet=Formulaire, Nom d'objet=Menu. Enregistrez et fermez.

ExécuterMacro (Ajouter Ami; ;)

FermerFenêtre (Formulaire; Menu; Avec confirmation)

L'action FermerFenêtre est nécessaire, sinon le formulaire Menu cache l'affichage correspondant à l'action qui nous intéresse. Mais, inversement, il faut revenir sur le formulaire Saisie des Amis pour que, à sa fermeture, on rouvre Menu.

5 - Ouvrez Saisie des Amis en mode création. Appelez la feuille des propriétés. Dans l'onglet Événement, ligne Sur fermeture, cliquez sur (...) et choisissez Générateur de macro. Installez l'action OuvrirFormulaire, Nom=Menu. Enregistrez et fermez la macro puis enregistrez et fermez le formulaire.

OuvrirFormulaire (Menu; Formulaire; ; ; ; Standard)

À la fermeture du formulaire, on repasse au Menu.

6 - Macro associée au bouton Anniversaires :

OuvrirFormulaire (Anniversaires; Formulaire; ; ; ; Standard)

FermerFenêtre (Formulaire; Menu; Avec confirmation)

Macro associée au bouton Téléphones :

OuvrirFormulaire (Téléphones des Amis; Formulaire; ; ; ; Standard)

FermerFenêtre (Formulaire; Menu; Avec confirmation)

7 - Revenez aux formulaires Anniversaires et Téléphones des amis en mode Création et créez les macros Sur fermeture :

OuvrirFormulaire (Menu; Formulaire; ; ; ; Standard)

8 - Il faut bien sûr faire de même pour le bouton Affichage. Le problème est que, s'il y a bien une action OuvrirTable, on ne peut attribuer d'événements à une table, donc pas de Sur fermeture. Par conséquent, vous devez d'abord créer un formulaire, qu'on va appeler F_Amis. Appelez 🗀*Créer-[Formulaires]-Plus de Formulaires-Feuille de données* en ayant la table Amis sélectionnée. Dans l'onglet Événement de la feuille de propriétés, ligne Sur fermeture, attribuez l'action :

OuvrirFormulaire (Menu; Formulaire; ; ; ; Standard)

Enregistrez, fermez.

9 - Au bouton Affichage, associez la macro suivante. Il faut que le paramètre Affichage soit Feuille de données.

OuvrirFormulaire (F_Amis; Feuille de données; ; ; ; Standard)

FermerFenêtre (Formulaire; Menu; Avec confirmation)

LA MACRO AUTOEXEC

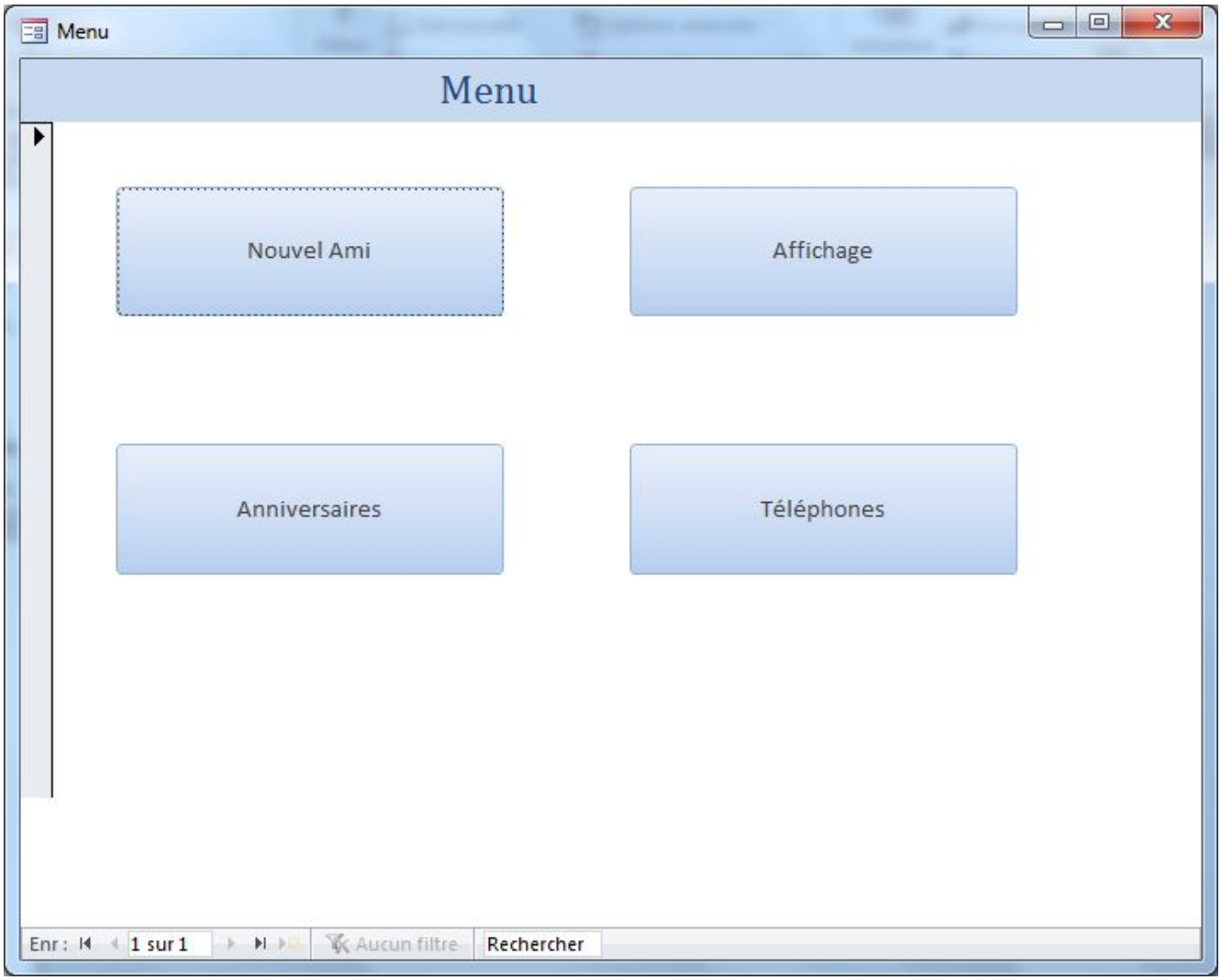
Si vous ouvrez le formulaire Menu, vous avez un comportement de menu. Vous pouvez perfectionner en fixant la propriété Fen indépendante à Oui dans l'onglet Autres de la fenêtre de propriétés.

Si la fenêtre est assez grande pour masquer pratiquement la fenêtre Access, on croit voir une application indépendante. Comment faire pour que cet aspect soit encore plus net ? Eh bien, ce qu'il faudrait, c'est que le menu apparaisse dès qu'on ouvre la base.

1 - Pour cela, créez une macro qui ouvre le formulaire Menu et enregistrez-la sous le nom AutoExec :

La macro AutoExec s'exécute dès que vous ouvrez la base, donc dès l'ouverture vous aurez le menu et vous vous croirez plongé dans l'application.

2 - Fermez la base et rouvrez-la : vous avez le menu.



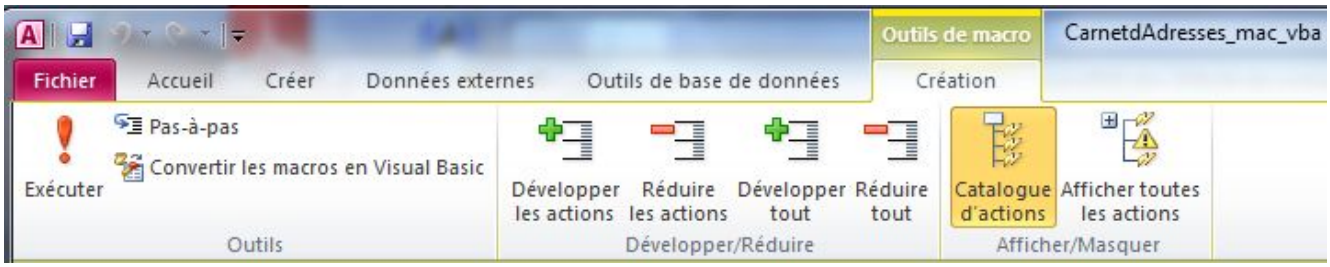
On n'a vu qu'une petite partie des possibilités des macros, et pourtant, ce n'est déjà pas mal. Eh bien, avec VBA, on aura des fonctionnalités beaucoup plus riches. La tendance actuelle est d'avoir recours à VBA plutôt qu'aux macros dès que l'application est un peu élaborée.

CONVERSION DE MACRO EN VBA

Il existe une commande qui permet de traduire une macro en procédure VBA.

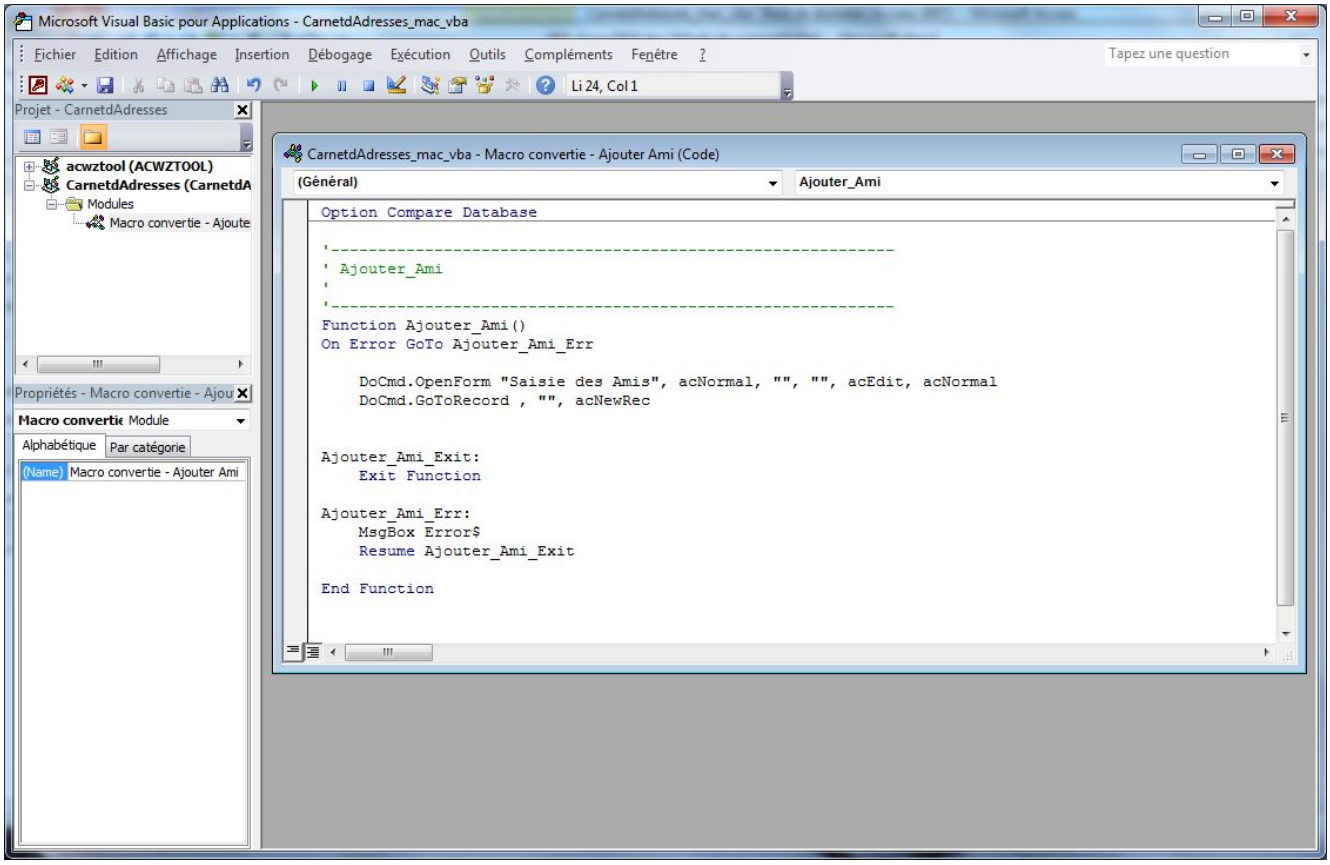
1 - Copiez la base CarnetdAdresses_mac.accdb en CarnetdAdresses_mac_vba.accdb et ouvrez cette nouvelle base.

2 - Cliquez droit sur la macro Ajouter Ami dans le volet de navigation et choisissez *Mode création* dans le menu déroulant. Le texte de la macro apparaît ainsi que l'onglet contextuel Outils de macro.



3 - Appelez 🗀*Outils de macro-[Outils]-Convertir les macros en Visual Basic*. Cliquez sur Convertir.

Après une boîte de message annonçant que la conversion est terminée, la fenêtre de l'éditeur VBA apparaît avec un module contenant la macro convertie.



EXAMINER LA TRADUCTION PRODUITE

Il est toujours instructif d'examiner la traduction produite. Dans l'exemple ci-dessus, seules les lignes `DoCmd`… traduisent les deux commandes de notre macro. Le `On Error` et le paragraphe

…_Err implantent le traitement d'erreur voulu car nous avons laissé la case *Ajouter une gestion des erreurs* cochée. Nous reviendrons sur les traitements d'erreurs.

Le principal intérêt de l'éditeur VBA est qu'il permet la modification du programme ou même son écriture à partir de rien. La traduction d'une macro peut servir de base de départ mais elle est limitée aux actions de la liste que nous avons vue. Pour des traitements élaborés, seul un programme VBA offre toutes les possibilités voulues.

Par macro, on ne peut que générer un programme à logique linéaire où toutes les actions se suivent en séquence ; on ne peut pas créer un programme où, en fonction de premiers résultats, on effectue telle action ou bien telle autre.

D'autre part, lorsqu'une sous-étape du traitement doit être répétée plusieurs fois, il faudra écrire les commandes autant de fois qu'il y a de répétitions, ce qui peut devenir infaisable s'il y en a des milliers. Ces possibilités appelées « alternatives » et « boucles » sont offertes par des instructions de VBA mais doivent être fournies directement. Ces instructions s'appellent instructions de structuration.

APPEL DIRECT DE L'ÉDITEUR VBA

Nous voyons maintenant d'autres manières d'appeler l'éditeur VBA. Vous avez le choix entre :

- 🗀*Créer-[Macros et code]-Visual basic* et
- 🗀*Outils de base de données-[Macro]-Visual Basic*.

Dans les deux cas, le raccourci-clavier est Alt+F11. Vous le retiendrez rapidement.

La fenêtre de l'éditeur VBA

On voit que l'éditeur VBA n'a pas adopté le ruban, mais a gardé l'interface par menus et barres d'outils ; ce sera peut-être pour une prochaine version.

La partie principale de la fenêtre est normalement occupée par les modules : toutes les instructions VBA doivent être écrites dans des procédures ou fonctions qui sont implantées dans des modules.

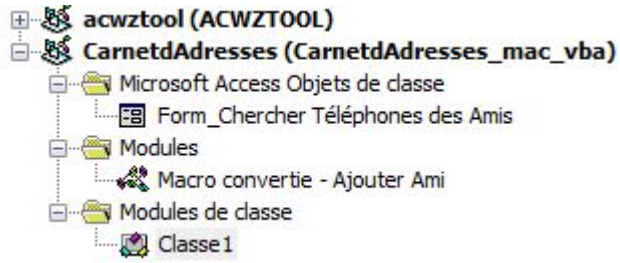
Le volet de gauche est normalement partagé entre l'Explorateur de projets et la feuille de propriétés. Si l'un de ces éléments n'apparaît pas, appelez le menu *Affichage* et cliquez sur l'élément voulu.

Parmi les autres éléments qu'on peut ainsi faire apparaître, l'Explorateur d'objets est très important. Les fenêtres Exécution, Variables locales, Espions et Pile des appels servent au dépannage des programmes.

ÉCRITURE DES INSTRUCTIONS VBA : L'ÉDITEUR VBA

CRÉER UN MODULE

Depuis l'écran Access, on arrive à l'écran VBA par la commande 🗀*Créer-[Macros et code]-Visual basic*, 🗀*Outils de base de données-[Macro]-Visual Basic* ou Alt+F11. On a vu dans la section précédente comment assurer que la fenêtre de projets soit présente. Elle a au moins une arborescence au nom de votre base de données ; celle-ci peut présenter les rubriques Microsoft Access Objets de classe, Modules et Modules de classe comme dans la figure (fictive) suivante :



- Si le programme que vous souhaitez écrire doit gérer la réponse à des événements concernant un formulaire ou un état, les modules correspondants apparaissent dans l'arborescence sous *Microsoft Access Objets de classe*. Pour créer le module, ouvrez le formulaire ou l'état en mode Création. Sélectionnez le contrôle concerné et affichez sa fenêtre de propriétés. Dans l'onglet *Événement*, cliquez sur le bouton (…) de la ligne « Sur événement voulu » et choisissez *Générateur de code*. La fenêtre de module apparaît.
- Dans les autres cas :
  - Sélectionnez le projet (clic sur sa ligne dans la fenêtre *Projets*).
  - Puis *Insertion – Module* pour un module normal. L'autre choix est *Module de classe*. Ce cas est traité dans un autre chapitre, donc plaçons-nous ici dans le cas du module normal.
  - Une fois le module créé, la rubrique *Modules* apparaît dans l'arborescence. Pour pouvoir écrire le programme, développez la rubrique, puis double-cliquez sur le nom du module voulu.
  - Il faut maintenant créer une procédure. Le menu *Insertion* a une rubrique *Procédure*, mais il suffit d'écrire `Sub <nom voulu>` dans le module.

SUPPRIMER UN MODULE

On peut avoir à supprimer un module, notamment parce qu'il est préférable de tout regrouper dans Module 1.

- Après avoir déplacé les procédures des autres modules dans Module 1, sélectionnez chaque module à supprimer par clic sur son nom sous la rubrique Modules.
- *Fichier – Supprimer Module 2* (le nom du module sélectionné apparaît dans le menu *Fichier*).
- Une BDi apparaît, proposant d'exporter le module. Cliquez sur Non.

EXPORTER/IMPORTER UN MODULE

Exporter :

Si dans la BDi précédente, vous cliquez sur Oui, vous exportez le module, c'est-à-dire que vous créez un fichier d'extension .bas qui contiendra le texte des procédures du module. Un tel fichier peut aussi se construire par :

- Mettez le curseur texte dans la fenêtre du module voulu.
- *Fichier – Exporter un fichier*.
- La BDi qui apparaît vous permet de choisir disque, répertoire et nom de fichier.

Importer :

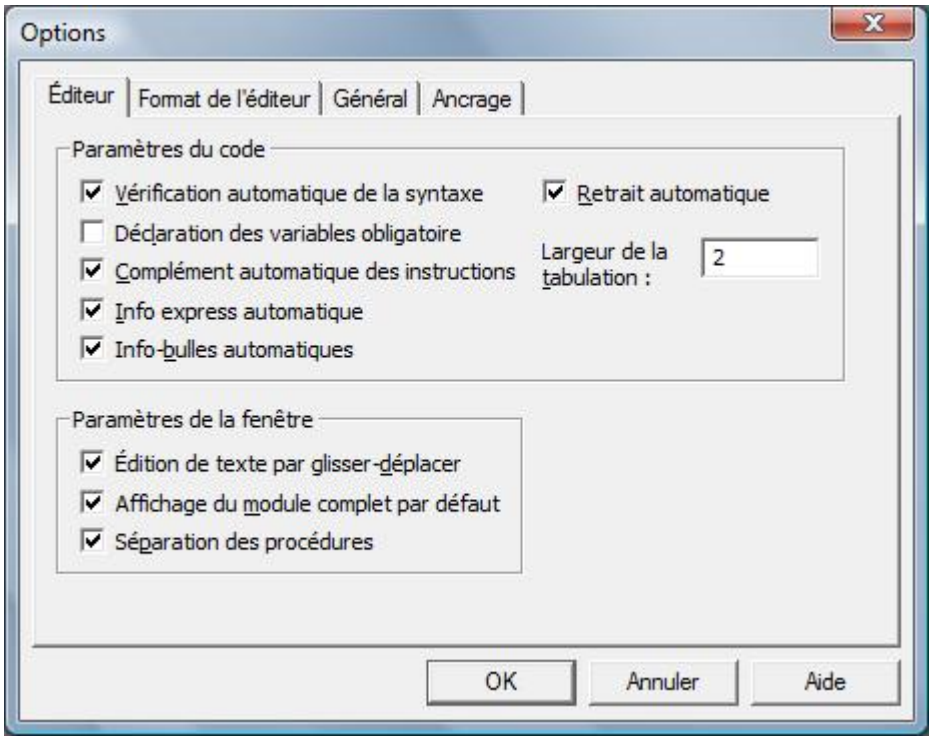
L'opération inverse est l'importation qui permet d'ajouter un fichier à un projet :

- Sélectionnez le projet concerné (par clic sur sa ligne dans la fenêtre de projets), puis faites *Fichier – Importer un fichier*.
- Dans la BDi, choisissez disque, répertoire et nom de fichier. Les extensions possibles sont .bas (module normal), et .cls (module de classe ou procédure événementielle associée à un formulaire ou un état).

Cette technique permet de développer des éléments, procédures ou objets servant pour plusieurs projets.

OPTIONS RÉGLANT LE FONCTIONNEMENT DE L'ÉDITEUR

Dans l'écran VBA, faites *Outils – Options*. Le fonctionnement de l'éditeur obéit aux onglets *Éditeur* et *Format de l'éditeur*. L'onglet *Éditeur* règle le comportement vis-à-vis du contenu du programme notamment les aides à l'écriture procurées par l'éditeur :



Les choix de la figure nous semblent les plus raisonnables.

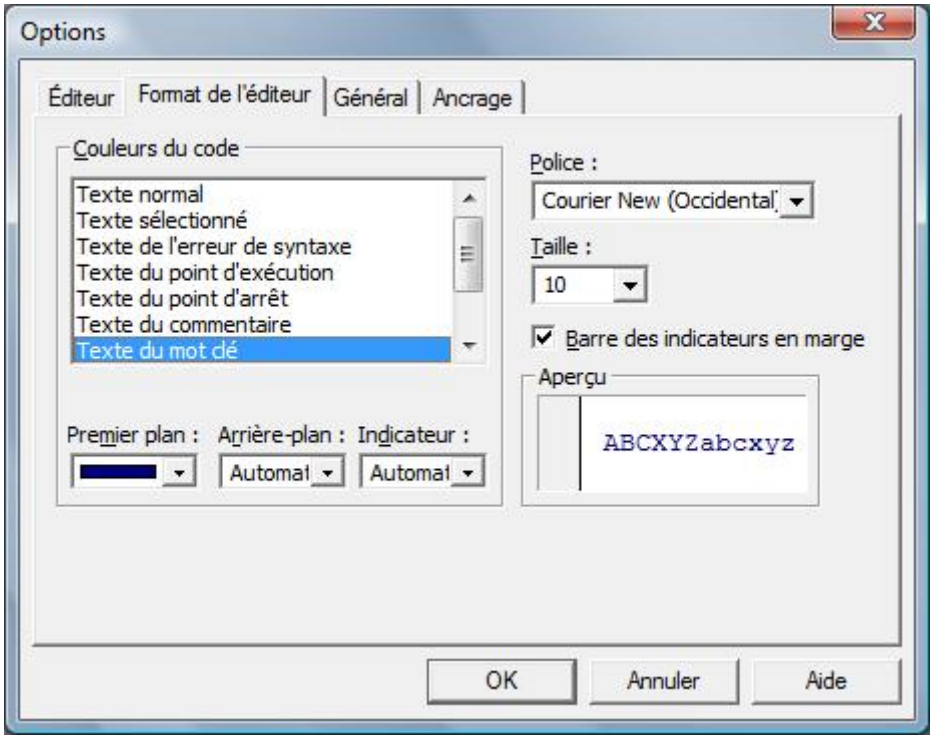
- ☑ *Vérification automatique de la syntaxe* parle d'elle-même.
- ☐ *Déclaration de variables obligatoire* si la case est cochée installe automatiquement Option Explicit en tête de tous les modules. Si la case n'est pas cochée, vous devez taper la directive partout où il le faut.
- ☑ *Complément automatique des instructions* présente les informations qui sont le complément logique de l'instruction au point où on est arrivé.
- ☑ *Info express automatique* affiche des informations au sujet des fonctions et de leurs paramètres au fur et à mesure de la saisie
- ☑ *Info-bulles automatiques* : en mode Arrêt, affiche la valeur de la variable sur laquelle le curseur est placé.
- ☑ *Retrait automatique* : si une ligne de code est mise en retrait, toutes les lignes suivantes sont automatiquement alignées par rapport à celle-ci. Pensez en même temps à choisir l'amplitude des retraits successifs (ci-dessus 2, au lieu de la valeur par défaut 4).

Les options *Paramètres de la fenêtre* sont moins cruciales.

- ☑ *Édition de texte par glisser-déplacer* permet de faire glisser des éléments au sein du code et de la fenêtre Code vers les fenêtres Exécution ou Espions.

- ☑ *Affichage du module complet par défaut* fait afficher toutes les procédures dans la fenêtre Code ; on peut, par moments, décider d'afficher les procédures une par une.
- ☑ *Séparation des procédures* permet d'afficher ou de masquer les barres séparatrices situées à la fin de chaque procédure dans la fenêtre Code. L'intérêt de cette option est diminué par le fait que ces séparations n'apparaissent pas à l'impression du listing ; une solution est d'insérer devant chaque procédure une ligne de commentaire remplie de tirets : `'--------…`

L'onglet *Format* de l'éditeur fixe les couleurs des différents éléments du code. C'est lui qui décide par défaut mots-clés en bleu, commentaires en vert, erreurs en rouge.



- ☑ *Barre des indicateurs en marge* affiche ou masque la barre des indicateurs en marge, indicateurs utiles pour le dépannage.
- Ayant choisi un des éléments dans la liste, vous déterminez la police, la taille et la couleur de façon classique ; en principe, on utilise une police de type Courrier parce qu'elle donne la même largeur à tous les caractères, mais rien ne vous y oblige.
- Les éléments possibles sont : Texte normal, Texte sélectionné, Texte de l'erreur de syntaxe, Texte du point d'exécution, Texte du point d'arrêt, Texte du commentaire, Texte du mot-clé, Texte de l'identificateur, Texte du signet, Texte de retour de l'appel.

RÈGLES FONDAMENTALES DE PRÉSENTATION

UNE INSTRUCTION PAR LIGNE

La règle fondamentale est d'écrire une instruction par ligne. Lorsque vous tapez sur la touche ↵, VBA suppose qu'on passe à la prochaine instruction. Cette règle admet deux exceptions qui n'interviennent que très rarement.

- On peut mettre plusieurs instructions sur une ligne à condition de les séparer par le caractère deux-points (:).

```
x = 3 : y = 5
```

Cette pratique est tout à fait déconseillée ; elle ne se justifie que pour deux instructions courtes formant en quelque sorte un bloc logique dans lequel il n'y aura en principe pas de risque d'avoir à insérer d'autres instructions.

- Une instruction peut déborder sur la (les) ligne(s) suivante(s). La présentation devient :

```
xxxxxxxxxxxxxxxxxxx(1ère      partie)xxxxxxxxxxxxxxxxxxx                    _
        yyyyyyy(2e partie)yyyyyyyyyyyyyyy
```

Les lignes, sauf la dernière, doivent se terminer par la séquence <espace><signe souligné>. Bien entendu, la coupure doit être placée judicieusement : là où l'instruction aurait naturellement un espace. On ne doit pas couper un mot-clé propre au langage, ni un nom de variable.

Cas particulier : on ne doit pas couper une chaîne de caractères entre guillemets (comme "Bonjour"). La solution est la suivante : on remplace la longue chaîne par une concaténation de deux parties ("partie 1" + "partie 2") et on coupera comme suit :

```
 ……."partie 1" +  _
"partie 2"  .
```

MAJUSCULES ET MINUSCULES

Sauf à l'intérieur d'une chaîne de caractères citée entre ", les majuscules et minuscules ne comptent pas en VBA. En fait, les mots-clés et les noms d'objets et de propriétés prédéfinis comportent des majuscules et minuscules et vous pouvez définir des noms de variables avec des majuscules où vous le souhaitez. Mais vous pouvez taper ces éléments en ne respectant pas les majuscules définies (mais il faut que les lettres soient les mêmes) : l'éditeur VBA rétablira automatiquement les majuscules de la définition ; pour les noms de variables, on se basera sur la 1re apparition de la variable (en principe sa déclaration).

Il en résulte un conseil très important : définissez des noms avec un certain nombre de majuscules bien placées et tapez tout en minuscules : si VBA ne rétablit pas de majuscules dans un nom, c'est qu'il y a une faute d'orthographe.

Un autre élément qui peut vous permettre de déceler une faute d'orthographe, mais seulement dans un mot-clé, est que si un mot n'est pas reconnu comme mot-clé, VBA ne l'affichera pas en bleu. Bien sûr, vous devez être vigilant sur ces points : plus tôt une faute est reconnue, moins il y a de temps perdu.

Pour les chaînes de caractères entre ", il s'agit de citations qui apparaîtront telles quelles, par exemple un message à afficher, le nom d'un client, etc. Il faut donc taper exactement les majuscules voulues.

COMMENTAIRES, LIGNES VIDES

Un commentaire est une portion de texte figurant dans le programme et n'ayant aucun effet sur celui-ci. La seule chose que VBA fait avec un commentaire, c'est de le mémoriser et de l'afficher dans le listing du programme. Les commentaires servent à donner des explications sur le programme, les choix de méthodes de traitement, les astuces utilisées, etc.

Ceci est utile pour modifier le programme, car, pour cela, il faut le comprendre ; c'est utile même pour le premier auteur du programme car lorsqu'on reprend un programme plusieurs mois après l'avoir écrit, on a oublié beaucoup de choses. Il est donc conseillé d'incorporer beaucoup de commentaires à un programme dès qu'il est un peu complexe.

VBA admet des commentaires en fin de ligne ou sur ligne entière.

En fin de ligne, le commentaire commence par une apostrophe. Ex. :

```
Remise = Montant * 0.1    ' On calcule une remise de 10%
```

Sur ligne entière, le commentaire commence par une apostrophe ou le mot-clé `Rem`. On utilise plutôt l'apostrophe. Si le commentaire occupe plusieurs lignes, chaque ligne doit avoir son apostrophe.

Les lignes vides sont autorisées en VBA ; elles peuvent servir à aérer le texte. Nous conseillons de mettre une apostrophe en tête pour montrer que le fait que la ligne soit vide est voulu par le programmeur.

LES ESPACES

Les espaces sont assez libres en VBA, mais pas totalement. Là où il peut et doit y avoir un espace, vous pouvez en mettre plusieurs, ou mettre une tabulation.

On ne doit en aucun cas incorporer d'espaces à l'intérieur d'un mot-clé, d'un nom d'objet prédéfini, d'un nombre ou d'un nom de variable : ces mots ne seraient pas reconnus.

Au contraire, pour former des mots, ces éléments doivent être entourés d'espaces, ou d'autres caractères séparateurs comme la virgule.

Les opérateurs doivent être entourés d'espaces, mais vous n'êtes pas obligé de les taper, l'éditeur VBA les fournira sauf pour &. Si vous tapez `a=b+c` vous obtiendrez `a = b + c`.

LES RETRAITS OU INDENTATIONS

Les instructions faisant partie d'une même séquence doivent normalement commencer au même niveau d'écartement par rapport à la marge. Lors de l'emploi d'instructions de structuration, les séquences qui en dépendent doivent être en retrait par rapport aux mots-clés de structuration. En cas de structures imbriquées, les retraits doivent s'ajouter. Exemple fictif :

```
x = 3
For I = 2 To 10
    a = 0.05 * I
    If b < x Then
         x = x - a
    Else
         b = b - a
    End If
Next I
```

En cas de nombreuses imbrications, le retrait peut être un peu grand : bornez-vous à 2 caractères à chaque niveau. Bien sûr, ces retraits ne sont pas demandés par le langage, ils n'ont que le but de faciliter la compréhension en faisant ressortir la structure du programme (ou plutôt, la structure souhaitée, car, dans son interprétation, VBA ne tient compte que des mots-clés, pas des indentations : mais justement un désaccord entre les mots-clés et les indentations peut vous aider à dépister une erreur).

Il est donc essentiel, bien que non obligatoire, que vous respectiez les indentations que nous suggérerons pour les instructions.

AIDE À LA RECHERCHE D'ERREURS

Nous avons vu plus haut que VBA introduisait de lui-même les majuscules voulues dans les mots-clés et les noms de variables, d'où notre conseil de tout taper en minuscules : s'il n'y a pas de transformation, c'est qu'il y a probablement une faute de frappe.

Pour les mots-clés, on a une aide supplémentaire : VBA met les mots-clés en bleu (en fait, la couleur choisie par option) ; si un mot n'est pas transformé, c'est qu'il n'est pas reconnu, donc qu'il y a une faute.

Une autre aide automatique est que, en cas d'erreur de syntaxe, VBA affiche aussitôt un message d'erreur et met l'instruction en rouge. Bien sûr cela ne décèle que les erreurs de syntaxe, pas les erreurs de logique du programme.

AIDES À L'ÉCRITURE

L'éditeur VBA complète automatiquement certaines instructions :

Dès que vous avez tapé une instruction `Sub` ou `Function`, VBA fournit le `End Sub` ou le `End Function`.

Si vous tapez `endif` sans espace, VBA corrige : `End If`. Attention, il ne le fait que pour celle-là : pour `End Select` ou pour `Exit Sub` ou d'autres, il faut taper l'espace.

Dès que vous tapez un espace après l'appel d'une procédure, ou la parenthèse ouvrante à l'appel d'une fonction, VBA vous suggère la liste des arguments. Il le fait toujours pour un élément prédéfini ; pour une procédure ou fonction définie par vous, il faut qu'elle ait été définie avant.

Dès que vous tapez le `As` dans une déclaration, VBA fournit une liste déroulante des types possibles ; il suffit de double-cliquer sur celui que vous voulez pour l'introduire dans votre instruction. Vous avancez rapidement dans la liste en tapant la première lettre souhaitée. Un avantage supplémentaire est qu'un élément ainsi écrit par VBA ne risque pas d'avoir de faute d'orthographe.

De même, dès que vous tapez le point après une désignation d'objet, VBA affiche la liste déroulante des sous-objets, propriétés et méthodes qui en dépendent et vous choisissez comme précédemment. L'intérêt est que la liste suggérée est exhaustive et peut donc vous faire penser à un élément que vous aviez oublié. Attention, cela n'apparaît que si l'aide en ligne est installée et si le type d'objet est connu complètement à l'écriture, donc pas pour une variable objet qui aurait été déclarée d'un type plus général que l'objet désigné (ex. `As Object`).

PROJETS, DIFFÉRENTES SORTES DE MODULES

DÉFINITION

Un **projet** est l'ensemble de ce qui forme la solution d'un problème (nous ne voulons pas dire « application » car ce terme a un autre sens, à savoir l'objet Application, c'est-à-dire Access lui-même), donc une base Access avec ses tables, ses formulaires *etc.* et tous les programmes écrits en VBA qui sont sauvegardés avec elle : le projet apparaît dans la fenêtre de projet sous le même nom que le fichier .accdb de la base. Les programmes sont dans des modules ; le texte des programmes est affiché dans des fenêtres de code. Il peut y avoir un module associé à chaque formulaire ou état. Il peut y avoir un certain nombre de modules généraux. De plus, le projet peut contenir aussi des modules de classe et des boîtes de dialogue créées par le programmeur : chaque BDi a en principe un module de code associé.

LES FENÊTRES DU PROJET

L'écran VBA contient principalement la fenêtre de projet où apparaît le projet associé à chaque fichier ouvert. Chaque projet y apparaît sous forme d'une arborescence (développable ou repliable) montrant tous les éléments du projet. Sous la fenêtre de projet peut apparaître une fenêtre Propriétés qui affiche les propriétés d'un élément choisi dans la fenêtre de projet.

La plus grande partie de l'écran sera consacrée aux fenêtres de code. Comme ces fenêtres sont en principe présentées en cascade, on choisit celle qui est en premier plan par clic dans le menu *Fenêtre*. On décide de l'affichage d'un tel élément par double-clic dans l'arborescence.

On peut faire apparaître d'autres fenêtres par clic dans le menu *Affichage*. C'est le cas des fenêtres de (l'Explorateur de) Projets, Propriétés, Explorateur d'objets, Exécution, Variables locales et Espions, ces trois dernières servant surtout au dépannage des programmes.

Le choix des fenêtres à afficher peut se faire aussi par des boutons de la barre d'outils Standard de l'écran VBA.

DIFFÉRENTES SORTES DE MODULES

À chacune des quatre rubriques de la hiérarchie dépendant du projet correspond une sorte de module. À *Microsoft Access Objets de classe* correspondent des modules où se trouveront les programmes de réponse aux événements d'un formulaire (ex. *Sur fermeture* d'un formulaire).

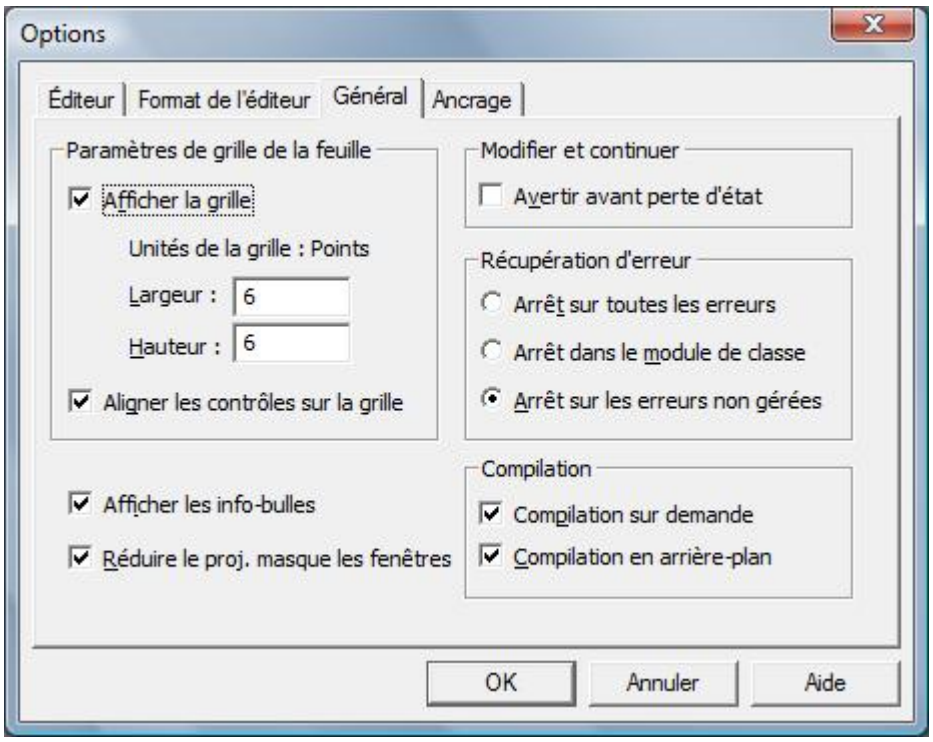
À *Modules* correspondent les différents modules « normaux » introduits. C'est dans ces modules (en principe, on les regroupe en un seul) que sont les procédures de calcul propres au problème.

La dernière sorte de modules dépend de la rubrique *Modules de classe* ; les modules de classe permettent de définir des objets propres au programmeur. Ils sont beaucoup moins souvent utilisés car, vu la richesse des objets prédéfinis en Access VBA, on en utilise rarement plus de 10%, alors on a d'autant moins de raisons d'en créer d'autres !

Une dernière rubrique, *Références* peut être présente dans l'arborescence, mais elle n'introduit pas de modules.

LA COMMANDE OUTILS-OPTIONS

Cette commande concerne les projets par ses onglets *Général* et *Ancrage*. L'onglet *Ancrage* décide quelles fenêtres vont pouvoir être ancrées c'est-à-dire fixées en périphérie de l'écran. Ce n'est pas vital. L'onglet *Général* a plus à dire :

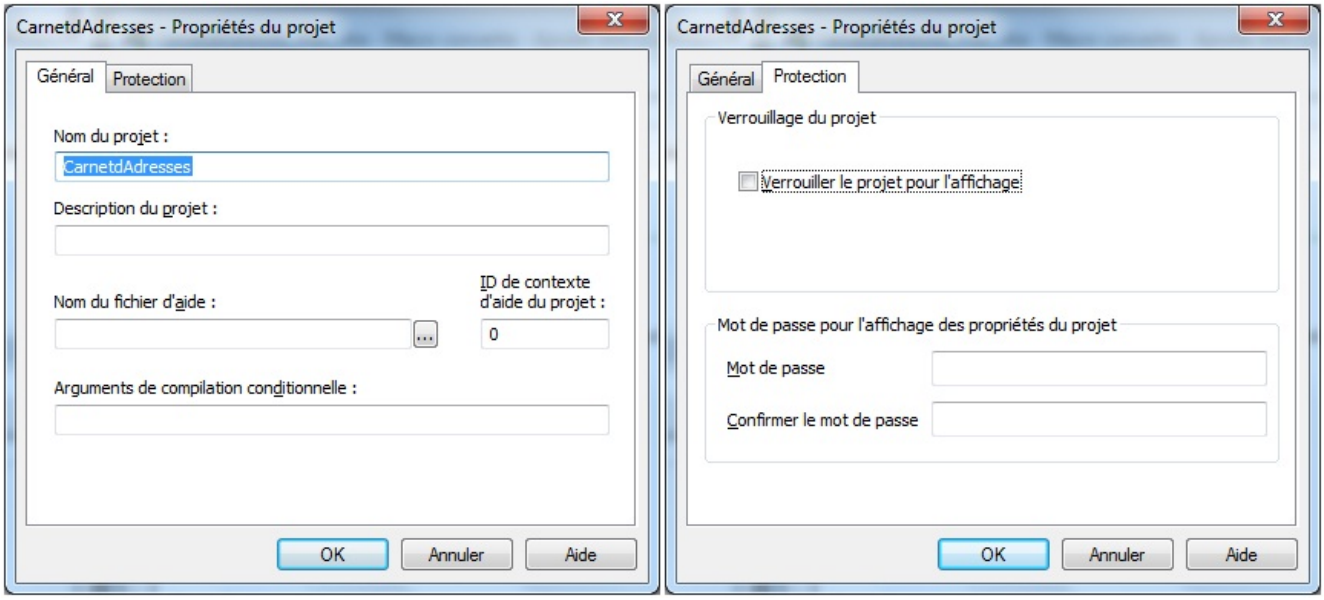


- ☑ *Afficher les info-bulles* affiche les info-bulles des boutons de barre d'outils.
- ☑ *Réduire le proj. masque les fenêtres* définit si les fenêtres de projet, UserForm, d'objet ou de module sont fermées automatiquement lors de la réduction du projet dans l'Explorateur de projet.
- Le cadre Modifier et continuer.
  - ☑ *Avertir avant perte d'état* active l'affichage d'un message lorsque l'action demandée va entraîner la réinitialisation de toutes les variables de niveau module dans le projet en cours.
- Le cadre Récupération d'erreur définit la gestion des erreurs dans l'environnement de développement Visual Basic. L'option s'applique à toutes les occurrences de Visual Basic lancées ultérieurement.
  - ⊙ *Arrêt sur toutes les erreurs* : en cas d'erreur quelle qu'elle soit, le projet passe en mode Arrêt.
  - ⊙ *Arrêt dans les modules de classe* : en cas d'erreur non gérée survenue dans un module de classe, le projet passe en mode Arrêt à la ligne de code du module de classe où s'est produite l'erreur.
  - ⊙ *Arrêt sur les erreurs non gérées* : si un gestionnaire d'erreurs est actif, l'erreur est interceptée sans passage en mode Arrêt. Si aucun gestionnaire d'erreurs n'est actif, le projet passe en mode Arrêt. Ceci est l'option la plus conseillée.

- Compilation
  - ☑ *Compilation sur demande* définit si un projet est entièrement compilé avant d'être exécuté ou si le code est compilé en fonction des besoins, ce qui permet à l'application de démarrer plus rapidement, mais retarde l'apparition des messages d'erreur éventuels dans une partie de programme rarement utilisée.
  - ☑ *Compilation en arrière-plan* définit si les périodes d'inactivité sont mises à profit durant l'exécution pour terminer la compilation du projet en arrière-plan, ce qui permet un gain de temps. Possible seulement en mode compilation sur demande.

LA COMMANDE OUTILS-PROPRIÉTÉS DE <NOM DU PROJET>

Cette commande fait apparaître une BDi avec deux onglets :



- L'onglet *Général* permet de donner un nom spécifique au projet, et surtout de fournir un petit texte descriptif. Les données concernant l'aide n'ont plus d'intérêt : la mode est maintenant de fournir une aide sous forme HTML. La compilation conditionnelle est sans réel intérêt.
- L'onglet *Protection* permet de protéger votre travail.
  - ☑ *Verrouiller le projet pour l'affichage* interdit toute modification de n'importe quel élément de votre projet. Il ne faut y faire appel que lorsque le projet est parfaitement au point !
  - La fourniture d'un mot de passe (il faut le donner deux fois, c'est classique) empêche de développer l'arborescence du projet dans la fenêtre Explorateur de projets si l'on ne donne pas le mot de passe. Donc un "indiscret" qui n'a pas le mot de passe n'a accès à aucune composante de votre projet.

LA COMMANDE OUTILS-RÉFÉRENCES

Permet de définir une référence à la bibliothèque d'objets d'une autre application pour y sélectionner des objets appartenant à cette application, afin de les utiliser dans votre code. C'est une façon d'enrichir votre projet.

LES DIFFÉRENTES SORTES D'INSTRUCTIONS

Les instructions VBA se répartissent en instructions exécutables ou ordres et instructions non exécutables ou déclarations.

INSTRUCTIONS EXÉCUTABLES

Ce sont les instructions qui font effectuer une action par l'ordinateur. Elles se répartissent en :

- **Instructions séquentielles**, telles que l'instruction qui sera exécutée après est l'instruction qui suit dans le texte.
  - La principale instruction de cette catégorie est *l'instruction d'affectation*, de la forme `[Set]<donnée>=<expression>` , où l'expression indique un calcul à faire. L'expression est calculée et le résultat est affecté à la donnée. En l'absence de `Set` (on devrait normalement mettre `Let`, mais il n'est jamais employé), l'expression conduit à une valeur et <donnée> est une variable ou une propriété d'objet ; elle reçoit la valeur calculée comme nouvelle valeur. Avec `Set`, l'expression a pour résultat un objet et <donnée> est une variable du type de cet objet : après l'instruction, cette variable permettra de désigner l'objet de façon abrégée. À part l'appel de procédures, cette instruction est la plus importante de tout le langage.
  - *Toute une série d'actions diverses*, notamment sur les fichiers (Open, Close, Print#...) ou sur certains objets (Load, Unload ...) ou encore certaines opérations système (Beep, Time…). Ces instructions pourraient d'ailleurs aussi bien être considérées comme des appels à des procédures ou des méthodes prédéfinies.

- **Instructions de structuration**, ou de rupture de séquence, qui rompent la suite purement linéaire des instructions, aiguillant le traitement vers une séquence ou une autre selon des conditions, ou faisant répéter une séquence selon les besoins. Ces instructions construisent donc la structure du programme. La plus importante est :
  - *L'appel de procédure* : on déroute l'exécution vers un bloc d'instructions nommé qui remplit un rôle déterminé. La fin de l'exécution de la procédure se réduit à un retour dans la procédure appelante juste après l'instruction d'appel. Cela permet de subdiviser un programme complexe en plusieurs petites unités beaucoup plus faciles à maîtriser. La plupart du temps, l'instruction se réduit à citer le nom de la procédure à appeler.

Les autres instructions de structuration permettent d'implémenter les deux structures de la programmation structurée.

- *La structure alternative* où, en fonction de certaines conditions, on fera une séquence ou bien une autre. VBA offre pour cela deux instructions principales, `If` qui construit une alternative à deux branches et `Select Case` qui permet plusieurs branches.
- *La structure itérative* ou **boucle,** où on répète une séquence jusqu'à ce qu'une condition soit remplie (ou tant que la condition contraire prévaut). VBA offre pour cette structure les instructions `Do…Loop…`, `While…Wend` et, surtout, `For…Next` qui est la plus employée.

INSTRUCTIONS NON EXÉCUTABLES OU DÉCLARATIONS

Ces instructions ne déclenchent pas d'actions de l'ordinateur, mais donnent des précisions au système VBA sur la manière dont il doit traiter les instructions exécutables. La plus importante de ces instructions est la déclaration de variable qui :

- annonce qu'on va utiliser une variable de tel ou tel nom,
- indique le type (par exemple réel, ou entier, etc.) de la variable, c'est-à-dire des données qu'elle va contenir. Il est évident que les calculs ne s'effectuent pas de la même façon sur un nombre entier ou sur un réel. C'est en cela que les déclarations orientent le travail de VBA. **Elles sont donc aussi importantes que les instructions exécutables.**

Place des déclarations de variables

Normalement, il suffit qu'une déclaration de variable soit n'importe où avant la première utilisation de cette variable. En fait on recommande vivement de placer les déclarations de variables en tête de leur procédure. Par ailleurs, certaines déclarations de variables doivent être placées en tête de module, avant la première procédure du module.

Parmi les déclarations importantes, les couples `Sub … End Sub` et `Function … End Function` délimitent respectivement une procédure ou une fonction. `Sub` et `Function` ont en outre le rôle de déclarer des éventuels arguments. Les deux `End …` sont à la fois des déclarations – elles délimitent la fin de la procédure ou de la fonction – et des instructions exécutables : lorsque l'on arrive sur elles on termine la procédure ou la fonction et on retourne à l'appelant.

DIRECTIVES

Les directives sont des déclarations particulières qui jouent un rôle global au niveau du projet. Elles sont placées tout à fait en tête de module. Certaines peuvent être spécifiées sous forme d'options de projet auquel cas la directive est écrite automatiquement en tête de tous les modules.

`Option Explicit`

Exige que toute variable soit déclarée. Nous conseillons vivement cette option car si vous faites une faute de frappe dans un nom de variable, en l'absence de cette option, VBA "croira" que vous introduisez une nouvelle variable, alors qu'avec cette option, il y aura un message d'erreur vous permettant de la corriger aussitôt.

`Option Base <0 ou 1>`

Fixe à 0 ou à 1 la première valeur des indices de tableaux. La valeur par défaut est 0. Souvent les programmeurs utilisent les indices à partir de 1 sans spécifier `Option Base 1` : l'élément 0 est laissé vide. Cette pratique a un inconvénient : si par erreur un indice était calculé à 0, la directive assurerait un message d'erreur.

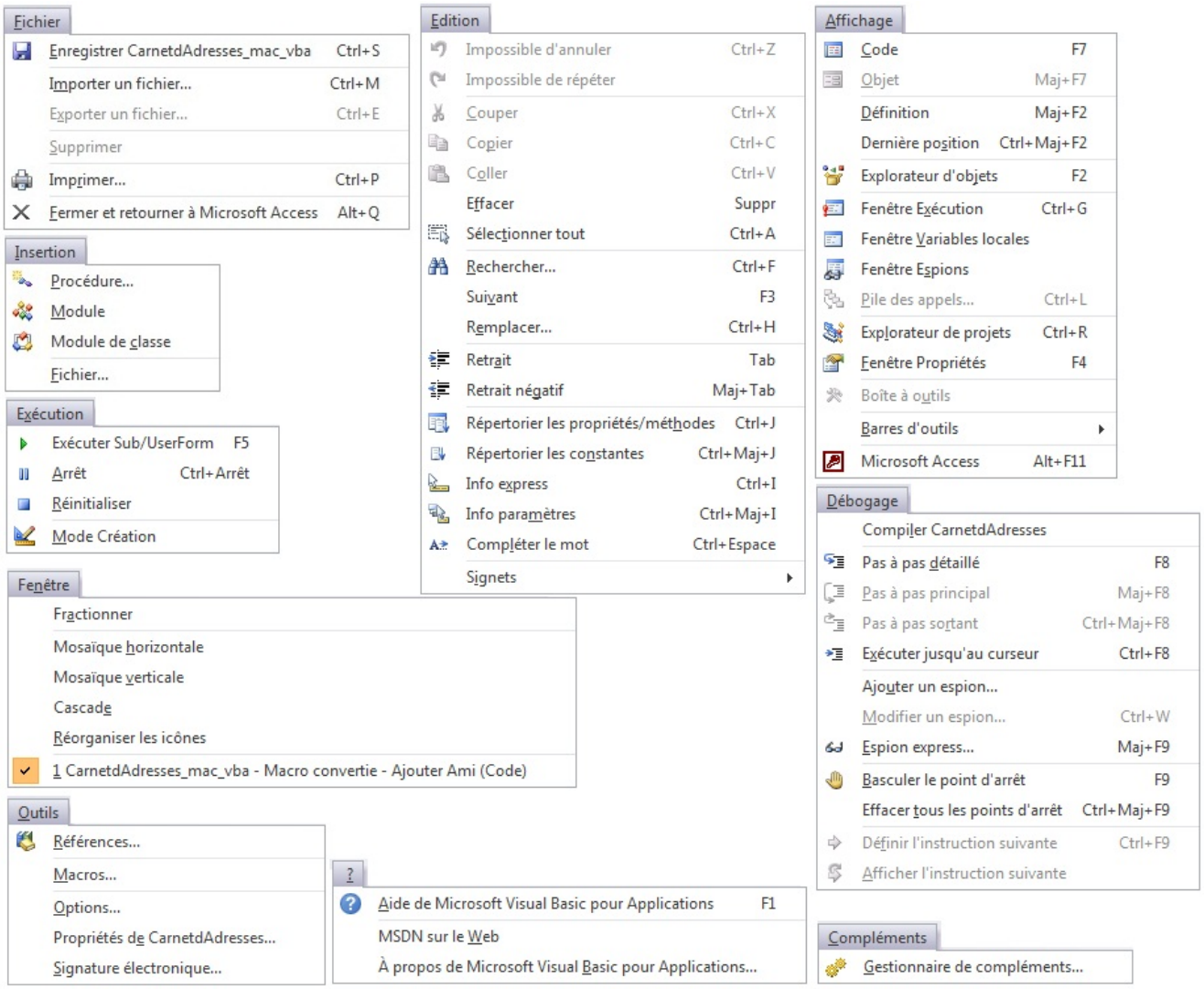
`Option Compare <choix>`

Fixe la façon dont les chaînes de caractères sont comparées. Avec `Text`, une majuscule et sa minuscule sont confondues alors qu'avec `Binary`, la comparaison est complète et les minuscules sont plus loin que les majuscules dans l'ordre alphabétique. Avec `Database`, on obéit aux paramètres régionaux de la base de données en cours.

`Option Private Module`

Déclare le module entier comme privé, donc aucun de ses éléments, variables, procédures ou fonctions ne sera accessible depuis un autre module.

LES MENUS DE L'ÉDITEUR VBA



N.B. Certaines rubriques peuvent varier légèrement en fonction du contexte, selon qu'on est dans une procédure ou non et selon ce qu'on a fait précédemment ; ainsi *Edition - Impossible d'annuler* peut devenir *Edition - Annuler*, etc.

Gestion d'une association

11

Étape 1 – Fichier HTM

Étape 2 – Nouveau membre

Étape 3 – Modification/Suppression

Pour aller plus loin

ÉTAPE 1 – FICHIER HTM

1. LE PROBLÈME

Nous allons gérer l'association des Amis des Animaux, c'est-à-dire inscrire les nouveaux membres, modifier leurs données, en supprimer, etc. Comme utilisation, nous allons créer la page WEB qui affichera le tableau des membres. D'autres utilisations sont envisageables, comme comptabiliser les cotisations, etc., nous les laissons de côté.

La base de données *AmisAnimaux.accdb* contiendra une seule table *Membres*. Dans la première étape, nous produisons le fichier .htm à partir de la BD telle qu'elle est. Les étapes suivantes feront évoluer la base. Dans l'état de départ *AmisAnimaux_00.accdb* du fichier que vous avez en téléchargement, il n'y a que la table Membres avec les données suivantes :

Membres

N°	Nom	Prénom	Adresse 1	Adresse 2	CP	Ville	Tel	eMail	Cotis à jour
1	DUCK	Donald	Le Bois Sacré	1 rue du Débarquement	14000	Caen	02 20 10 15 02		☐
2	DUPONT	Georges	Ker Mag	20 Av Joffre	44500	La Baule	02 40 60 20 00	gdupont@mail.fr	☐
3	DURAND	Charles	12 rue de la Lune		75003	Paris	06 03 89 78 81	cdurand@yahoo.com	☐
4	FRICOTIN	Bibi	2 rue de Bellevue		75019	Paris			☐
5	GUIGNOL	Albert	13 Traboule Dogue		69001	Lyon	04 78 25 00 00		☐
6	MOUSE	Mickey	Impasse du Fromage		38000	Grenoble		mmouse@noos.fr	☐
✳	#####								☐

La rubrique <Cotis. à jour> est prévue, mais elle ne sera pas gérée ici. A part N° qui est en entier à incrémentation automatique, tous les champs sont en texte.

Pour obtenir le fichier *AmisAnimaux_0.accdb*, vous créez le formulaire nommé *Menu*, qui propose un bouton par fonctionnalité comme suggéré au chapitre 10 :

On voit sur la figure qu'après l'explication succincte à côté de chaque bouton, figure le nom de la procédure associée dans Module1 ; pour le moment ces procédures sont toutes vides. Voici le module de classe du formulaire :

```
Private Sub B_HTM_Click()
  GenerHTM
End Sub

Private Sub B_ModifSupp_Click()
  ModifMembre
End Sub

Private Sub B_NouvMemb_Click()
  NouvMembre
End Sub

Private Sub B_Quit_Click()
  DoCmd.Close acForm, "Menu", acSaveNo
  DoCmd.Quit acQuitSaveAll
End Sub
```

Utilisation des fichiers téléchargés

Chaque cas pratique a son dossier. Les fichiers des cas pratiques ont leur nom terminé par le n° d'étape. Pour reproduire le passage de l'étape n à n+1, vous copiez le fichier n dans un nouveau répertoire, ainsi que les fichiers auxiliaires éventuels. Ensuite, copiez le fichier n sous le nom n+1 et faites les manipulations sur ce fichier n+1.

2. ROUTINE D'INITIALISATION

Nous commençons par l'introduction d'une routine d'initialisation `InitGen` et de quelques variables: `Chem` (le dossier des fichiers : au départ, le fichier .htm sera dans le même que la base de données), `Sep` (le séparateur \ ou : - il faut changer l'instruction Sep = … sur Mac et c'est probablement la seule à changer), `NbRub` (nombre de rubriques traitées : on le diminue de 1 car on ne s'occupe pas de la cotisation ❹) et le tableau `NomsRub` (les noms de rubriques). On introduit une variable `Cnx` pour la connexion, `Rst` pour le `Recordset` de la table des membres et `NbLig` le nombre d'enregistrements. Celui-ci est déterminé par `RecordCount` ❷ car le `Recordset` est ouvert en lecture seule ❶.

Les routines qui suivent doivent être saisies dans le module *Module 1*.

Voici la routine `InitGen` précédée des déclarations.

```
'--------Version 1.0------------
Public Chem As String, Sep As String, Rub As String
Public Cnx As ADODB.Connection, Rst As ADODB.Recordset
Public NbRub As Integer, NomsRub(10) As String, FF As Object
Public NbLig As Integer, Kol As Integer, Lig As Integer

Sub InitGen()
  Sep = "\" ' Remplacer par Sep=":" sur Mac
  Chem = CurrentProject.Path + Sep
  Set Cnx = CurrentProject.Connection
  Set Rst = New ADODB.Recordset
  Rst.Open "Membres", Cnx, adOpenStatic, adLockReadOnly ❶
  NbLig = Rst.RecordCount ❷
  NbRub = 0
  For Each FF In Rst.Fields ❸
    NbRub = NbRub + 1
    NomsRub(NbRub) = FF.Name
  Next
  NbRub = NbRub - 1 ❹
End Sub
```

En ❸, on détermine les noms de rubrique par balayage de la collection `Fields`.

3. CONSTRUCTION DU FICHIER .HTM

La structure est très simple : début de la page Web, tableau des membres, fin de la page. Le tableau a lui-même un début et une fin entourant une double structure répétitive pour les lignes (les membres) et les colonnes (les rubriques). Voici le texte HTML représentant le tableau à afficher :

```
<html><head>                                |début de la page
<title>Les Amis des Animaux</title>                 |
</head><body><center>                       |
<h2>Membres de l'Association</h2>                   |
<h2>Les Amis des Animaux</h2></center>              |
<table border width=95%>                            |     |début du tableau
<tr><td>nom de rubrique …</td></tr>                  |     |     |ligne d'en-tête
<tr>                                        |     |     |ligne
<td>rubrique</td>                           |     |     |     |rubrique
<td>rubrique</td>                           |     |     |     |rubrique
</tr>                                       |     |     |fin de la ligne
</table>                                            |     |fin du tableau
</body></html>                              |fin de la page
```

Voici la page Web correspondant au fichier *Membres.htm* obtenu à partir des données page 170 grâce à la procédure `GenerHTM` ci-après.

Membres de l'Association

Les Amis des Animaux

N°	Nom	Prénom	Adresse 1	Adresse 2	CP	Ville	Tel	eMail
1	DUCK	Donald	Le Bois Sacré	1 rue du Débarquement	14000	Caen	02 20 10 15 02	
2	DUPONT	Georges	Ker Mag	20 Av Joffre	44500	La Baule	02 40 60 20 00	eMail
3	DURAND	Charles	12 rue de la Lune		75003	Paris	06 03 89 78 81	eMail
4	FRICOTIN	Bibi	2 rue de Bellevue		75019	Paris		
5	GUIGNOL	Albert	13 Traboule Dogue		69001	Lyon	04 78 25 00 00	
7	MOUSE	Mickey	Impasse du Fromage		38000	Grenoble		eMail

```
'-------------------------------------------------GenerHTM--
Sub GenerHTM()
  InitGen
  Open Chem + "Membres.htm" For Output As #1
  Print #1, "<html><head>" + vbCr;  ' Début page
  Print #1, "<title>Les Amis des Animaux</title>" + vbCr;
  Print #1, "</head><body><center>" + vbCr;
  Print #1, "<h2>Membres de l'Association</h2>" + vbCr;
  Print #1, "<h2>Les Amis des Animaux</h2></center>" + vbCr;
  Print #1, "<table border width=95%>" + vbCr; ' Début tableau
  Print #1, "<tr>" + vbCr;
  For Kol = 1 To NbRub                ' Noms de rubriques
    Print #1, "<td><b>" + NomsRub(Kol) + "</b>" + "</td>" + vbCr;
  Next Kol
  Print #1, "</tr>" + vbCr;
```

```
  Rst.MoveFirst
  For Lig = 1 To NbLig
    Print #1, "<tr>" + vbCr;
    For Kol = 1 To NbRub - 1          'Les rubriques
      Rub = CStr(Nz(Rst.Fields(NomsRub(Kol)).Value, ""))
      If Rub = "" Then Rub = " "
      Print #1, "<td>" + Rub + "</td>" + vbCr;
    Next Kol
    Rub = CStr(Nz(Rst.Fields(NomsRub(NbRub)).Value, "")) 'eMail à part
    If Rub = "" Then Rub = " " Else _
         Rub = "<a href=""mailto:" + Rub + """>eMail</a>"
    Print #1, "<td>" + Rub + "</td>" + vbCr;
    Print #1, "</tr>" + vbCr;
    Rst.MoveNext
  Next Lig
  Print #1, "</table>" + vbCr;
  Print #1, "</body></html>" + vbCr;
  Close 1
  Rst.Close: Cnx.Close
  Set Rst = Nothing: Set Cnx = Nothing
End Sub
```

La procédure GenerHTM commence par appeler InitGen. Ensuite, chaque ligne d'écriture html se fait par un print # de la chaîne de caractères voulue ; on termine par vbCr et **;** pour avoir un parfait contrôle des lignes. Si la rubrique est vide, on met " " (l'espace en HTML) pour assurer la continuité de la bordure. Pour la rubrique eMail, la chaîne est :

`"<a href=""mailto:adresse"">eMail</a>"` : remarquez les doubles guillemets pour incorporer un guillemet.

On utilise les variables `Lig` (numéro de ligne), `Kol` (numéro de rubrique/colonne) et `Rub` (le texte de la rubrique). Celui-ci est obtenu dans l'enregistrement courant du Recordset à partir du champ correspondant au nom de rubrique.

Le `<b> </b>` entourant l'écriture de chaque nom de rubrique, le met en gras.

Le nom du fichier Web produit est fixé à *Membres.htm* dans l'instruction `Open`. Une telle chose est en principe à éviter : on doit paramétrer au maximum. Dans notre exemple, on pourrait introduire une variable `NomFichWeb` obtenue par une InputBox :

```
NomFichWeb=InputBox("Nom du fichier Web à créer ? ",,"Membres.htm")
```

Nous vous laissons l'implantation complète à titre d'exercice complémentaire.

Notez aussi dans ce contexte qu'on ne se préoccupe pas de savoir s'il existe déjà un fichier .htm de même nom dans le dossier : si c'est le cas, ce fichier est écrasé et remplacé par le fichier que vous créez, sans qu'il y ait le moindre message pour vous avertir. Dans ce cas, le fichier .htm reflètera l'état actuel de la table Membres au moment de l'exécution de la commande et c'est probablement ce qui est souhaité. Si vous voulez conserver un état antérieur de la page Web, il faut préalablement sauvegarder le fichier *Membres.htm*.

Après l'ouverture du fichier, une batterie de `Print #` écrit les lignes de début du fichier .htm tel qu'esquissé page 200. On implante la balise de début de tableau et la boucle `For Kol`…. remplit la 1re ligne avec les noms de rubrique.

La boucle `For Lig = …` parcourt les enregistrements du `Recordset` (Méthodes `MoveFirst` au départ, puis `MoveNext` dans la boucle). Pour chaque enregistrement d'un membre de l'association, il y aura une ligne du tableau, donc on implante la balise `<tr>` de début de ligne. On a ensuite la boucle sur les rubriques. On termine par la balise `</tr>` de fin de ligne.

Les rubriques sont traitées en deux temps : les `NbRub-1` premières rubriques sont traitées dans la boucle `For Col`… Le contenu trouvé sur la feuille est converti en chaîne. S'il est vide, on inscrira ` ` qui figure un espace : si on ne le faisait pas, on aurait une case vide dans le tableau et la bordure aurait une discontinuité inesthétique (ceci est un problème HTML qui sort du sujet de ce livre). Bien sûr, chaque rubrique est annoncée par la balise <td>. Remarquez l'emploi de `Nz` car une rubrique non renseignée dans la table donne la valeur `Null`.

La dernière rubrique est traitée à part car il n'y a pas qu'à recopier l'adresse eMail, il faut construire le lien en insérant le contenu lu sur la feuille des membres entre les balises `<a>` et `</a>`.

Le programme se termine par les balises de fin de tableau et de fin d'HTML et, surtout, par la fermeture du fichier à ne pas oublier sous peine d'écriture incomplète.

Une fois la frappe finie, faites *Débogage – Compiler...*. Un certain nombre d'erreurs peuvent vous être signalées : comparez avec le listing ci-dessus et corrigez. Sauvegardez la base de données (sous le nom *AmisAnimaux_1.accdb*).

Attention, vous ne devez pas écraser le fichier de même nom téléchargé. Vous devez avoir conservé une copie des fichiers originaux téléchargés dans un autre dossier.

Il vous reste à tester l'exécution, ce qui s'obtient en cliquant sur le bouton Génère HTM de la feuille *Menu*. Si vous n'avez pas fait d'erreur, vous devriez obtenir un fichier *Membres.htm* et la visualisation par votre navigateur doit avoir l'aspect de la figure de la page 172.

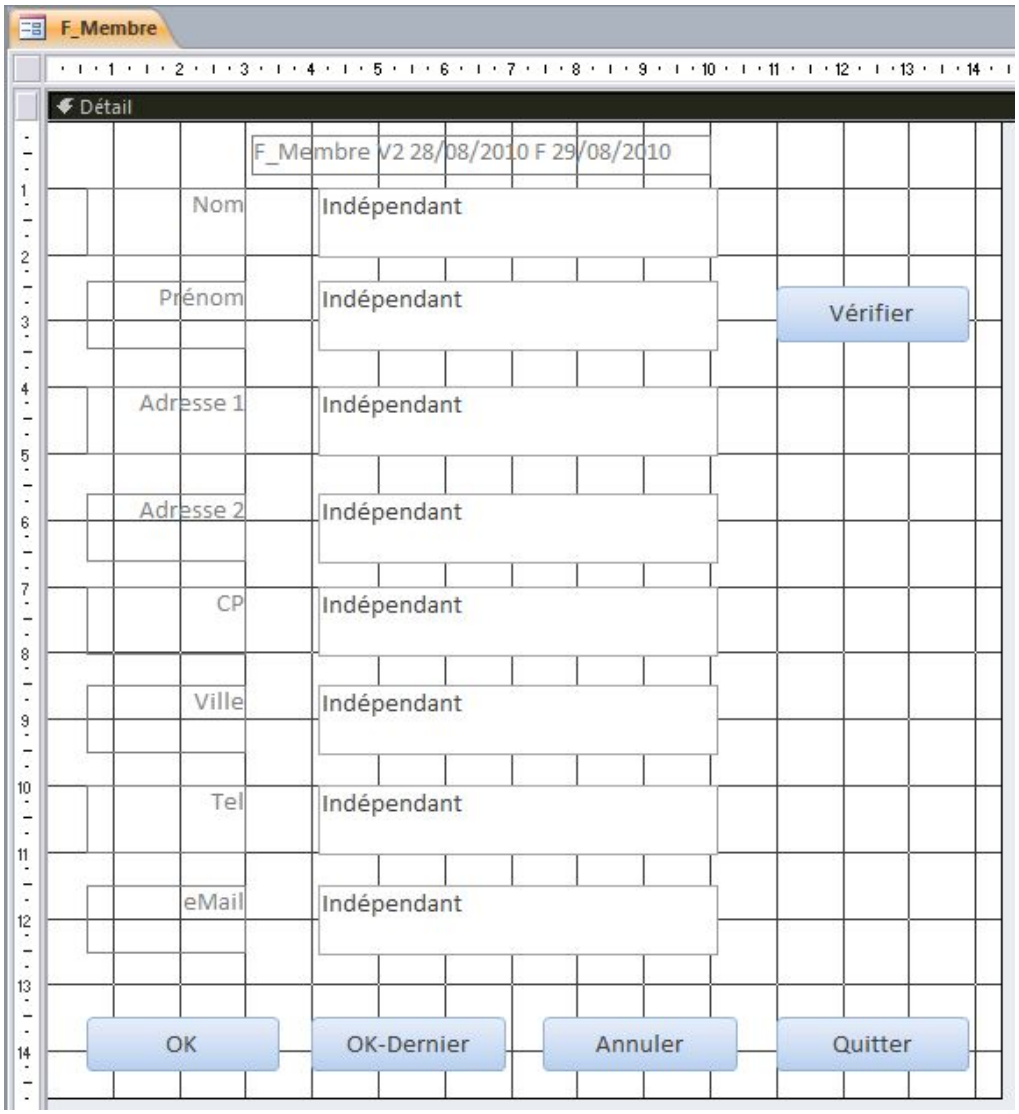
Remarquez que nous précédons chaque procédure d'un commentaire formé de tirets qui servira de séparateur si vous imprimez le listing. En effet, à l'impression, les traits de séparation installés par l'Editeur VBA n'apparaissent pas. Le rappel du nom de la procédure à la fin aide à la repérer si le listing est très long.

ÉTAPE 2 – NOUVEAU MEMBRE

1. CRÉER UN FORMULAIRE

Nous passons à la gestion de la base de données, et, d'abord, à l'entrée d'un nouveau membre. Il nous faut donc un formulaire pour entrer ses données.

- Copiez la base *AmisAnimaux_1. accdb* en *AmisAnimaux_2.accdb* sur laquelle vous allez travailler.
- Passez en mode création de formulaire ; nommez-le F_Membre, Caption Nouveau Membre. Fixez les propriétés Fen Indépendante, Fen Modale à Oui, Boutons de déplacement à Non.
- Créez une TextBox avec son Label à côté ; donnez la valeur « Droite » à la propriété Aligner le texte du label ; sélectionnez les deux et faites Copier.
- Faites Coller 7 fois : vous avez 8 couples (on gère 8 rubriques).
- Donnez aux labels les `Captions` respectives *Nom*, *Prénom* … (les noms de rubriques) ; donnez aux TextBox les noms TB_Nom *etc.* (les noms des rubriques).
- Créez 5 boutons, les 4 premiers en bas, le 5e à côté du prénom. Donnez les Name (et Caption) respectifs B_OK (OK), B_OKDern (OK-Dernier), B_Annul (Annuler), B_Quit (Quitter) et B-Ver (Vérifier).
- Créez un Label en haut avec `Visible` = `False` et la `Caption = F_Membre V2 date F date` : ce label apparaîtra au listing alors que le titre du formulaire n'apparaît pas. Le formulaire doit avoir l'aspect :



Le bouton Vérifier devra être cliqué après avoir entré nom et prénom : le système préviendra si nom et prénom identiques se trouvent déjà dans la base. Les boutons de validation ne seront activés qu'après cette vérification. La dualité OK, Annuler /OK Dernier, Quitter permet d'entrer une série de membres : pour le dernier, on valide par OK-Dernier.

Cette gestion utilise deux booléens `Satisf` et `Dernier` : `Satisf` est vrai si on a validé les données d'un membre, `Dernier` si c'est le dernier de la série. On a en plus une variable `Mode` qui distinguera le cas Nouveau membre du cas Modification, car, par économie, nous utiliserons le même formulaire, à peine modifié. Ces variables sont publiques, ainsi que le tableau `DonMemb` des données du membre qui sera géré comme objet `Scripting.Dictionary`.

En résumé, il s'ajoute en tête du Module 1 les déclarations :

```
Public Mode As Integer, Satisf As Boolean, Dernier As Boolean, DonMemb As Object
```

2. MODULE DE CLASSE DU FORMULAIRE

Ce module est essentiellement formé des procédures événements des contrôles du formulaire, mais il peut s'ajouter d'autres procédures si, comme ce sera le cas ici, une même opération est à effectuer à partir de plusieurs contrôles.

Pour implanter une procédure événement, vous sélectionnez le contrôle puis cliquez sur la ligne Sur événement dans l'onglet Événements la fenêtre de propriétés ; choisissez Générateur de code. Il s'implante souvent inopinément des routines `_Click`, laissées vides : pensez à les supprimer.

Le module a deux variables globales au niveau module Nm et Pr qui contiendront le nom et le prénom du membre en cours d'ajout.

Nous avons d'abord la routine `Form_Current` où nous n'implantons que la branche Mode=0 : on ne fait qu'initialiser Nm et Pr et fixer le titre du formulaire et la légende du bouton « Vérifier ».

Lorsqu'on a entré un nom et/ou un prénom, on désactive les boutons « OK », car on doit effectuer la vérification, d'où les deux routines `TB_Nom_Exit` et `TB_Prénom_Exit`. Elles prennent en compte respectivement le nom et le prénom entrés.

Les quatre routines des boutons de validation `B_Annul_Click`, `B_OK_Click`, `B_OKDern_Click` et `B_Quit_Click` sont très semblables : avant de fermer le formulaire elles fixent en conséquence les booléens sur lesquels est basée la gestion : `Dernier` est mis à vrai pour les boutons qui terminent une série B_OKDern et B_Quitter. `Satisf` est mis à faux par les boutons d'annulation et à vrai par les boutons OK.

Les boutons « OK » appellent la procédure `CaptureDon` qui transfère les données des contrôles dans le tableau `DonMemb`. En effet, si on clique sur « OK », c'est que les données entrées dans les contrôles sont correctes. Le tableau Dictionnaire `DonMemb` sert à les mémoriser pour récupération dans Module 1 ; il est géré comme vu au chapitre 10 dans la section sur le dictionnaire de données. Mais avant, d'appeler `CaptureDon`, les routines font appel à la fonction booléenne `VerNomPren` qui prend la valeur `False` s'il manque soit le nom soit le prénom : dans ce cas, inutile de capturer les données et on ne ferme pas le formulaire.

La routine `B_Ver_Click` est la plus délicate. Pour le moment, nous n'implantons que la branche Mode=0. Pour vérifier s'il y a déjà un membre ayant à la fois même nom et même prénom, on crée un objet commande qui exécutera une requête de comptage des enregistrements satisfaisant à cette condition. Le résultat de la requête est Res(0). Value.

Si ce résultat est différent de 0, on demande à l'utilisateur s'il veut tout de même entrer ce membre (deux membres peuvent avoir mêmes nom et prénom ; espérons qu'ils n'ont pas la même adresse !) et alors on active aussi les boutons OK. Si la réponse est non, l'utilisateur doit changer le nom et/ou le prénom ou bien annuler.

```
Dim Nm As String, Pr As String

Sub CaptureDon()
  Dim NomTB As String, ct As Control, Tx As String
  For Each ct In Controls
    If Left(ct.Name, 3) = "TB_" Then
      NomTB = Mid(ct.Name, 4)
      ct.SetFocus
      Tx = Nz(ct.Text, "")
      DonMemb.Add NomTB, Tx
    End If
  Next
End Sub

Function VerNomPren() As Boolean
  If (Nm = "") Or (Pr = "") Then
    MsgBox "Il faut un nom et un prénom"
    VerNomPren = False
  Else
    VerNomPren = True
  End If
End Function

Private Sub B_Annul_Click()
  Dernier = False
  Satisf = False
  DoCmd.Close acForm, Me.Name, acSaveNo
End Sub

Private Sub B_OK_Click()
  If Not VerNomPren Then Exit Sub
  CaptureDon
  Dernier = False
  Satisf = True
  DoCmd.Close acForm, Me.Name, acSaveNo
End Sub

Private Sub B_OKDern_Click()
  If Not VerNomPren Then Exit Sub
  CaptureDon
  Dernier = True
  Satisf = True
  DoCmd.Close acForm, Me.Name, acSaveNo
End Sub

Private Sub B_Quit_Click()
  Dernier = True
  Satisf = False
  DoCmd.Close acForm, Me.Name, acSaveNo
End Sub
```

```
Private Sub B_Ver_Click()
  Dim Cm As New ADODB.Command, Res, Rep
  If Mode = 0 Then
    Set Cm.ActiveConnection = CurrentProject.Connection
    Cm.CommandType = adCmdText
    Cm.CommandText = "select count(Nom) from Membres " + _
        "where (Nom='" + Nm + "') and (Prénom='" + Pr + "')"
    Set Res = Cm.Execute
    If Res(0).Value <> 0 Then
      Rep = MsgBox("Ce nom et prénom sont déjà présents" + vbCr + _
        "voulez-vous tout de même entrer ce membre", vbYesNo + _
        vbExclamation)
      If Rep = vbYes Then B_OK.Enabled = True: B_OKDern.Enabled = True
    Else
      B_OK.Enabled = True: B_OKDern.Enabled = True
    End If
  Else
  ' laissé vide pour le moment
  End If
End Sub

Private Sub Form_Current()
  Nm = "": Pr = ""
  If Mode = 0 Then
    Me.Caption = "Nouveau membre"
    B_Ver.Caption = "Vérifier"
  Else
  ' laissé vide pour le moment
  End If
End Sub

Private Sub TB_Nom_Exit(Cancel As Integer)
  Nm = Nz(TB_Nom.Text, "")
  B_OK.Enabled = False
  B_OKDern.Enabled = False
End Sub

Private Sub TB_Prénom_Exit(Cancel As Integer)
  Pr = Nz(TB_Prénom.Text, "")
  B_OK.Enabled = False
  B_OKDern.Enabled = False
End Sub
```

3. PROCÉDURE NOUVMEMBRE DANS MODULE 1

```
'-------------------------------------------------NouvMembre--
Sub NouvMembre()
  Dim NomC As String, ff As Object
  Dernier = False
  While Not Dernier
    Set DonMemb = CreateObject("Scripting.Dictionary")
    Set Cnx = CurrentProject.Connection
    Set Rst = New ADODB.Recordset
    Rst.Open "Membres", Cnx, adOpenDynamic, adLockOptimistic
    Satisf = False
```

```
    Mode = 0
    DoCmd.OpenForm "F_Membre", , , , acFormAdd, acDialog
    If Satisf Then
      Rst.AddNew
      For Each ff In Rst.Fields
        NomC = ff.Name
        If (NomC <> "N°") And (NomC <> "Cotis à jour") _
            Then ff.Value = DonMemb(NomC)
      Next
      Rst.Update
    End If
    DonMemb.RemoveAll: Set DonMemb = Nothing
    Rst.Close: Cnx.Close
    Set Rst = Nothing: Set Cnx = Nothing
  Wend
End Sub
```

La structure de `NouvMemb` est en fait simple :

```
Dernier=False
While Not Dernier        'Tant qu'on n'a pas entré le dernier de la série
|  Set DonMemb..    'Préparer le dictionnaire des données
|  Rst.Open…        'Ouvrir le Recordset
|     DoCmd.OpenForm     'Afficher le formulaire
|      If Satisf Then           'Si on a obtenu une donnée correcte
|      |    Rst.AddNew   'Crée nouvel enregistrement
|   |    For Each ff     'Remplissage des champs
|      |    |…
|      |    Next
|      |    Rst.Update   'Installe le nouvel enregistrement
|      End If
|   'Fait le ménage en vue d'un éventuel enregistrement suivant
Wend
```

La boucle `For Each ff` suit exactement le schéma de remplissage des champs vu au chapitre 10 puisque les TextBox ont été nommées suivant le modèle TB_<nom du champ>.

Sauvegardez le fichier sous le nom *AmisAnimaux_2.accdb* avec toujours la même précaution d'avoir conservé une copie intacte des fichiers originaux téléchargés.

Pour tester le programme à l'étape 2, vous cliquez sur le bouton « Nouveau membre », vous entrez une série de nouveaux membres (clic sur OK après chaque et sur OK-Dernier après le dernier) : vous devez vérifier que les données des membres sont bien entrées.

4. QUELQUES AMÉLIORATIONS

Nous proposons deux légers changements dans le code qui vont servir en vue de la 3e étape. Cela va constituer le fichier *AmisAnimaux_2a.accdb*. Dans `NouvMembre`, la boucle de remplissage des champs `For Each ff` va servir plusieurs fois. Donc nous l'implantons dans une procédure `RemplitChamps` et la partie concernée du module est maintenant :

```
'-----------------------------------------------------RemplitChamps--
Sub RemplitChamps()
  Dim NomC As String, ff As Object
  For Each ff In Rst.Fields
    NomC = ff.Name
    If (NomC <> "N°") And (NomC <> "Cotis à jour") _
            Then ff.Value = DonMemb(NomC)
    Next
End Sub

'-----------------------------------------------------NouvMembre--
Sub NouvMembre()
  Dernier = False
  While Not Dernier
    Set DonMemb = CreateObject("Scripting.Dictionary")
    Set Cnx = CurrentProject.Connection
    Set Rst = New ADODB.Recordset
    Rst.Open "Membres", Cnx, adOpenDynamic, adLockOptimistic
    Satisf = False
    Mode = 0
    DoCmd.OpenForm "F_Membre", , , , acFormAdd, acDialog
    If Satisf Then
      Rst.AddNew
      RemplitChamps
      Rst.Update
    End If
    DonMemb.RemoveAll: Set DonMemb = Nothing
    Rst.Close: Cnx.Close
    Set Rst = Nothing: Set Cnx = Nothing
  Wend
End Sub
```

De même, dans `B_Ver_Click` du module associé au formulaire, la partie de test de la présence du nom et prénom est susceptible de généralisation. Nous créons une fonction booléenne `TestSQL` qui a le texte d'une requête comme argument et qui renvoie vrai s'il existe des enregistrements satisfaisant à la requête.

```
Function TestSQL(chSQL As String) As Boolean
  Dim Cm As New ADODB.Command, Res
  Set Cm.ActiveConnection = CurrentProject.Connection
  Cm.CommandType = adCmdText
  Cm.CommandText = chSQL
  Set Res = Cm.Execute
  If Res(0).Value <> 0 Then TestSQL = True Else TestSQL = False
  Set Cm = Nothing
End Function

Private Sub B_Ver_Click()
  Dim Ch As String, Rep
  If Mode = 0 Then
    Ch = "select count(Nom) from Membres " + _
        "where (Nom='" + Nm + "') and (Prénom='" + Pr + "')"
```

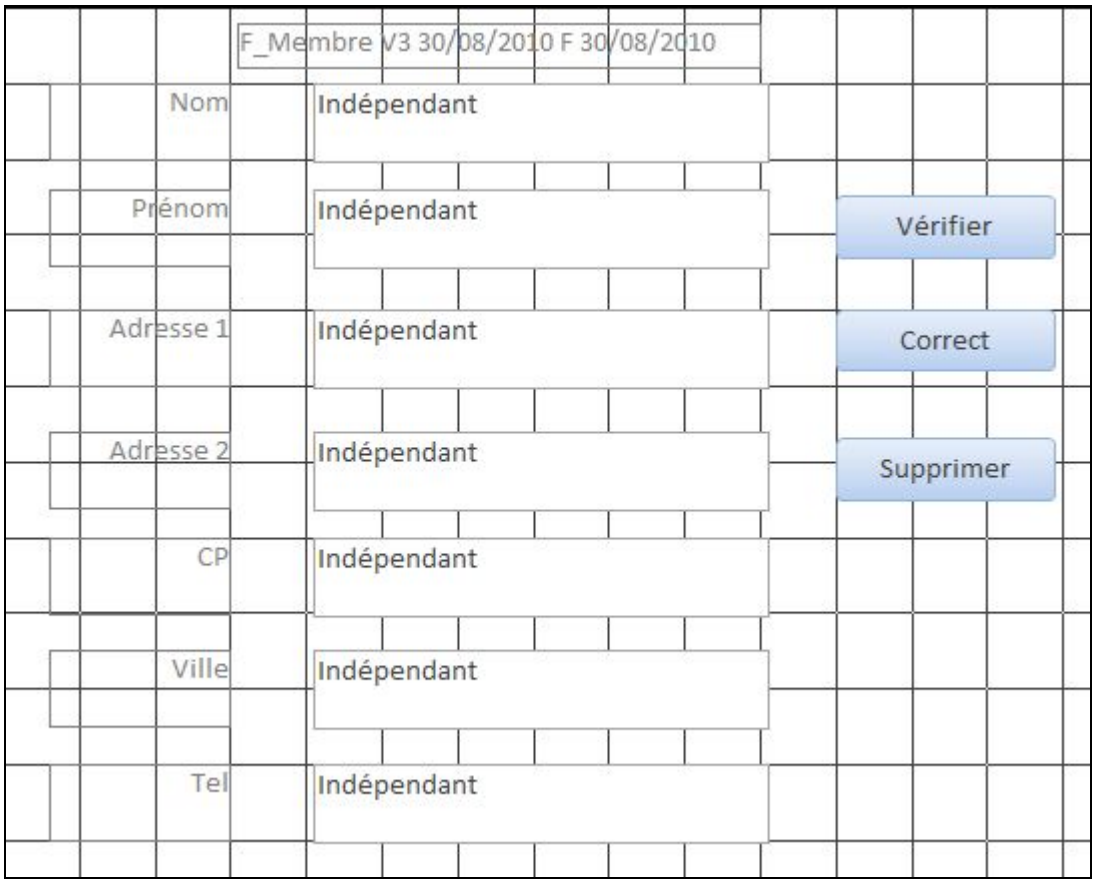
```
    If TestSQL(Ch) Then
      Rep = MsgBox("Ce nom et prénom sont déjà présents" + vbCr + _
        "voulez-vous tout de même entrer ce membre", vbYesNo + _
        vbExclamation)
      If Rep = vbYes Then B_OK.Enabled = True: B_OKDern.Enabled = True
    Else
      B_OK.Enabled = True: B_OKDern.Enabled = True
    End If
  Else
  ' laissé vide pour le moment
  End If
End Sub
```

ÉTAPE 3 – MODIFICATION/SUPPRESSION

1. CONSTRUIRE LE FORMULAIRE

Copiez *AmisAnimaux_2a.accdb* en *AmisAnimaux_3.accdb* sur lequel vous allez travailler.

Pour la modification, le problème est de trouver l'enregistrement à modifier. On fait la recherche sur le nom : lorsqu'on a trouvé une concordance, on affiche l'ensemble des données de l'enregistrement et l'utilisateur doit cliquer sur Correct si c'est l'enregistrement cherché. Sinon, il doit cliquer sur Chercher/Suivant car il peut y avoir plusieurs membres de même nom. Le libellé « Chercher/Suivant » remplace le libellé « Vérifier » ; les deux boutons supplémentaires sont visibles et actifs seulement si Mode=1. Dans ce cas, on change aussi le titre du formulaire dans la routine `Form_Current`. Voici le nouvel aspect du formulaire :



Nous essayons d'avoir un mode d'emploi assez perfectionné pour la recherche de l'enregistrement à modifier. Si vous fournissez le prénom, la recherche se fait sur nom et prénom exacts. Si vous ne fournissez pas le prénom, on accepte tout nom contenant ce que vous avez tapé dans la zone nom.

2. PROCÉDURES DU MODULE DE CLASSE

Dans la fenêtre de code associée au formulaire, vous avez un certain nombre de routines à modifier, et il s'ajoute les routines de clic des deux boutons supplémentaires.

Le module de classe a une nouvelle variable, `EnCours`, vraie si ce n'est pas la 1re fois que vous tapez sur le bouton : la 1re fois, il signifie Chercher, les fois suivantes, il signifie Suivant..

Les routines des quatre boutons de validation ainsi que les deux routines d'Exit des deux TextBox sont inchangées mais `CaptureDon` commence par un vidage du dictionnaire : en effet, maintenant, pour une ouverture du formulaire, il peut y avoir plusieurs enregistrements à regarder lors de la recherche de l'enregistrement à modifier.

`CaptureDon` est complétée par un certain nombre de routines de transfert entre TextBox et le dictionnaire `DonMemb` ou entre champs de l'enregistrement en cours : `VideBDi`, `RemplitBDi`, `RecupChamps` et `RemplitChamps`. Les deux dernières sont dans Module1.

La procédure `Form_Current` a maintenant aussi la branche pour Mode=1 (sous le `Else`). Même la branche Mode = 0 est à modifier puisqu'il s'ajoute la gestion des activations et visibilité des boutons supplémentaires *B_Correct* et *B_Suppr*. Dans la branche Mode=1 on initialise `EnCours` à `False`.

La routine `B_Correct_Click` active le bouton Supprimer et les deux « OK » puisque le membre sur lequel on veut agir est maintenant trouvé.

`B_Suppr_Click` demande une confirmation et, si oui, effectue la suppression. On agit sur l'enregistrement en cours. Remarquez que, après la suppression de l'enregistrement, on appelle `B_Annul_Click` : en effet, au retour dans le programme appelant, tout doit se passer comme si on avait annulé car la modification de la table Membres a été effectuée.

C'est la routine `B_Ver_Click` qui subit les plus importantes modifications. La branche Mode=0 est inchangée, ce qui prouve la solidité de notre programmation. La branche `Else` est subdivisée en deux selon la valeur de `EnCours`. Si `EnCours` vaut 0, on commence une recherche : la chaîne de requête est construite en fonction de la présence ou non du prénom (si le nom est absent, il y a un message d'erreur). On teste alors s'il y a des enregistrements conformes et à ce moment on ouvre le Recordset sur la requête proprement dite.

Si `EnCours` vaut 1, c'est que l'utilisateur a cliqué sur Chercher/Suivant : il faut donc chercher plus loin à condition qu'on ne dépasse pas la fin du `Recordset` ; dans ce cas, un message en avertit l'utilisateur. Si on a pu lire un enregistrement, les appels à `RecupChamps` puis `RemplitBDi` affichent les données : l'utilisateur peut juger s'il doit cliquer sur Correct ou demander l'aller plus loin.

Si on clique sur Correct, les données pourront être modifiées et les valeurs modifiées validées puisque les boutons de validation auront été activés par `B_Correct_Click`.

```
Dim Nm As String, Pr As String
Dim EnCours As Boolean

Sub CaptureDon()
  Dim NomTB As String, ct As Control, Tx As String
  DonMemb.RemoveAll
  For Each ct In Controls
    If Left(ct.Name, 3) = "TB_" Then
      NomTB = Mid(ct.Name, 4)
      ct.SetFocus
      Tx = Nz(ct.Text, "")
      DonMemb.Add NomTB, Tx
    End If
  Next
End Sub

Sub VideBDi()
  Dim NomTB As String, ct As Control
  DonMemb.RemoveAll
  For Each ct In Controls
    If Left(ct.Name, 3) = "TB_" Then
      NomTB = Mid(ct.Name, 4)
      ct.SetFocus
      ct.Text = ""
    End If
  Next
End Sub
```

```
Sub RemplitBDi()
  Dim NomTB As String, ct As Control
  For Each ct In Controls
    If Left(ct.Name, 3) = "TB_" Then
      NomTB = Mid(ct.Name, 4)
      ct.SetFocus
      ct.Text = Nz(DonMemb(NomTB), "")
    End If
  Next
End Sub

Function VerNomPren() As Boolean
  If (Nm = "") Or (Pr = "") Then
    MsgBox "Il faut un nom et un prénom"
    VerNomPren = False
  Else
    VerNomPren = True
  End If
End Function

Private Sub B_Annul_Click()
  Dernier = False
  Satisf = False
  DoCmd.Close acForm, Me.Name, acSaveNo
End Sub

Private Sub B_Correct_Click()
  B_Suppr.Enabled = True
  B_OK.Enabled = True
  B_OKDern.Enabled = True
End Sub

Private Sub B_OK_Click()
  If Not VerNomPren Then Exit Sub
  CaptureDon
  Dernier = False
  Satisf = True
  DoCmd.Close acForm, Me.Name, acSaveNo
End Sub

Private Sub B_OKDern_Click()
  If Not VerNomPren Then Exit Sub
  CaptureDon
  Dernier = True
  Satisf = True
  DoCmd.Close acForm, Me.Name, acSaveNo
End Sub

Private Sub B_Quit_Click()
  Dernier = True
  Satisf = False
  DoCmd.Close acForm, Me.Name, acSaveNo
End Sub
```

```
Private Sub B_Suppr_Click()
  Dim Rep
  Rep = MsgBox("Etes-vous sûr de vouloir supprimer ce membre ? ", _
    vbYesNo + vbQuestion)
  If Rep = vbYes Then
    Rst.Delete adAffectCurrent
    Rst.Update
    B_Annul_Click
  End If
End Sub

Function TestSQL(chSQL As String) As Boolean
  Dim Cm As New ADODB.Command, Res
  Set Cm.ActiveConnection = CurrentProject.Connection
  Cm.CommandType = adCmdText
  Cm.CommandText = chSQL
  Set Res = Cm.Execute
  If Res(0).Value <> 0 Then TestSQL = True Else TestSQL = False
  Set Cm = Nothing
End Function

Private Sub B_Ver_Click()
  Dim Ch As String, Rep
  If Mode = 0 Then
    Ch = "select count(Nom) from Membres " + _
        "where (Nom='" + Nm + "') and (Prénom='" + Pr + "')"
    If TestSQL(Ch) Then
      Rep = MsgBox("Ce nom et prénom sont déjà présents" + vbCr + _
        "voulez-vous tout de même entrer ce membre", vbYesNo + _
        vbExclamation)
      If Rep = vbYes Then B_OK.Enabled = True: B_OKDern.Enabled = True
    Else
      B_OK.Enabled = True: B_OKDern.Enabled = True
    End If
  Else                          'Mode
    If Not EnCours Then
      If Nm = "" Then
        MsgBox "Il faut un nom."
        Exit Sub
      End If
      Ch = "select count(Nom) from Membres where "
      If Pr = "" Then
        Ch = Ch + "InStr(Nom, '" + Nm + "')>0"
      Else
        Ch = Ch + "(Nom='" + Nm + "') and (Prénom='" + Pr + "')"
      End If
      If Not TestSQL(Ch) Then
        MsgBox "Pas d'enregistrement conforme. Changez ou annulez."
        Exit Sub
      End If
      EnCours = True
      Ch = Replace(Ch, "count(Nom)", "*")
      Rst.Open Ch, Cnx, adOpenDynamic, adLockOptimistic
      Rst.MoveFirst
    Else                'Not EnCours
      Rst.MoveNext
```

```
      If Rst.EOF Then
        MsgBox "Plus d'enregistrements."
        EnCours = False
        VideBDi
        Rst.Close
        Exit Sub
      End If
    End If                'Not EnCours
    RecupChamps
    RemplitBDi
  End If                  'Mode
End Sub

Private Sub Form_Current()
  Nm = "": Pr = ""
  If Mode = 0 Then
    Me.Caption = "Nouveau membre"
    B_Ver.Caption = "Vérifier"
    B_Correct.Enabled = False
    B_Correct.Visible = False
    B_Suppr.Enabled = False
    B_Suppr.Visible = False
  Else
    Me.Caption = "Modification/Suppression membre"
    B_Ver.Caption = "Chercher/Suivant"
    B_Correct.Enabled = True
    B_Correct.Visible = True
    B_Suppr.Enabled = False
    B_Suppr.Visible = True
    EnCours = False
  End If
End Sub

Private Sub TB_Nom_Exit(Cancel As Integer)
  Nm = Nz(TB_Nom.Text, "")
  B_OK.Enabled = False
  B_OKDern.Enabled = False
End Sub

Private Sub TB_Prénom_Exit(Cancel As Integer)
  Pr = Nz(TB_Prénom.Text, "")
  B_OK.Enabled = False
  B_OKDern.Enabled = False
End Sub
```

3. PROCÉDURE MODIFMEMBRE ET ANNEXES

```
'-----------------------------------------------------RecupChamps--
Sub RecupChamps()
  Dim NomC As String, ff As Object
  DonMemb.RemoveAll
  For Each ff In Rst.Fields
    NomC = ff.Name
    If (NomC <> "N°") And (NomC <> "Cotis à jour") _
            Then DonMemb.Add NomC, ff.Value
    Next
End Sub
```

```
'-----------------------------------------------------RemplitChamps--
Sub RemplitChamps()
  Dim NomC As String, ff As Object
  For Each ff In Rst.Fields
    NomC = ff.Name
    If (NomC <> "N°") And (NomC <> "Cotis à jour") _
            Then ff.Value = DonMemb(NomC)
    Next
End Sub

'-----------------------------------------------------ModifMembre--
Sub ModifMembre()
  Dernier = False
  While Not Dernier
    Set DonMemb = CreateObject("Scripting.Dictionary")
    Set Cnx = CurrentProject.Connection
    Set Rst = New ADODB.Recordset
    Satisf = False
    Mode = 1
    DoCmd.OpenForm "F_Membre", , , , acFormEdit, acDialog
    If Satisf Then
      RemplitChamps
      Rst.Update
    End If
    DonMemb.RemoveAll: Set DonMemb = Nothing
    If Rst.State <> 0 Then Rst.Close
    Cnx.Close
    Set Rst = Nothing: Set Cnx = Nothing
  Wend
End Sub
```

La structure est semblable à `NouvMemb` mais certaines instructions comme l'ouverture du Recordset ne sont plus là : elles sont dans le module de classe car la recherche de l'enregistrement à modifier ou supprimer est en plusieurs épisodes séparés par clic sur Suivant. La différence importante est la valeur donnée à la variable `Mode` : une erreur ou l'oubli de cette instruction empêcherait le fonctionnement correct. Notez aussi que la fermeture du Recordset est conditionnelle : en effet, il peut être fermé par B_Ver_Click et si on essaie de le fermer une 2e fois, on se « paie » un message d'erreur.

Nous sommes parvenus au fichier *AmisAnimaux_3.accdb*. Pour essayer quelques modifications, vous cliquez sur le bouton et modifiez quelques données. Ensuite, vous fermez le formulaire Menu et ouvrez la table Membres pour vérifier que les modifications sont bien entrées et sont à la bonne place.

Voici quelques directions de possibles améliorations :

Offrir un système d'Aide

Offrir un système d'aide. Il faut bien sûr créer les fichiers .htm voulus ; ce n'est pas le sujet de ce livre. Ensuite, il faut fournir au moins un bouton dans le formulaire *Menu* un bouton dans le formulaire *F_Membre*. Nous avons vu comment écrire les routines de clic de ces boutons.

Gérer des rubriques supplémentaires

C'est simple en s'inspirant des routines écrites pour les rubriques en place.

Ajouter des fonctionnalités

Par exemple faire un système de relance des adhérents en retard de cotisation ; il faudrait ajouter la rubrique date de dernière cotisation… C'est utile pour toutes les associations.

Améliorer l'ergonomie

Si on clique sur Chercher/Suivant une fois de trop, le nom ne sera pas trouvé et il faudra reprendre la recherche au début. Il serait plus ergonomique d'implanter un bouton Précédent permettant des allers et retours.

Maintenant, quelques leçons à retenir de cette étude de cas (et de toutes) :

- Complémentarité de tous les éléments d'un projet : les instructions sont écrites en fonction de la structure des données dans les tables ; les procédures événements liés aux contrôles de BDi et les appels des BDi sont écrits en fonction les uns des autres et les transmissions de données doivent être prévues… Autre comportement de la BDi, autre façon de l'utiliser.
- Par ailleurs, le fait qu'on puisse ouvrir les tables directement sous Access offre un moyen indépendant de notre programme de les examiner et donc de vérifier ce que fait notre programme. Bien sûr, cet accès est surtout important pendant la phase de mise au point.