Prototypage

Développement Rapide d'Applications avec Microsoft Access

Licence Professionnelle d'Informatique
Option Génie Logiciel

2002/2003

Pierre Gérard
IUT de Villetaneuse
pierre.gerard@iutv.univ-paris13.fr

Version originale du cours : Freddy Didier

TD1

Personnaliser une application Access avec VBA

Objectifs

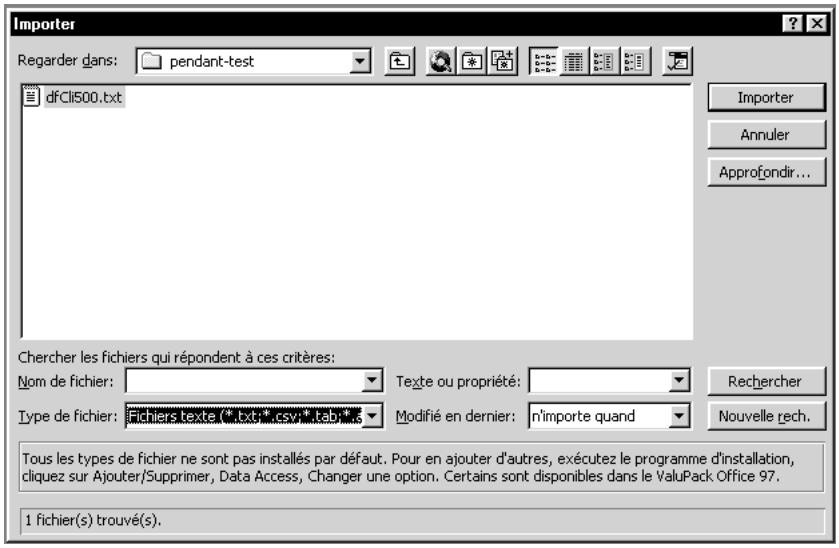
1. **Savoir importer des données externes**
2. **Savoir modifier la structure des données d'une table**
3. **Savoir réaliser un formulaire avec l'assistant**
4. **Savoir créer un bouton avec l'assistant**
5. **Savoir afficher et comprendre de code VB créé par l'assistant**
6. **Savoir créer un bouton sans l'assistant**

Début

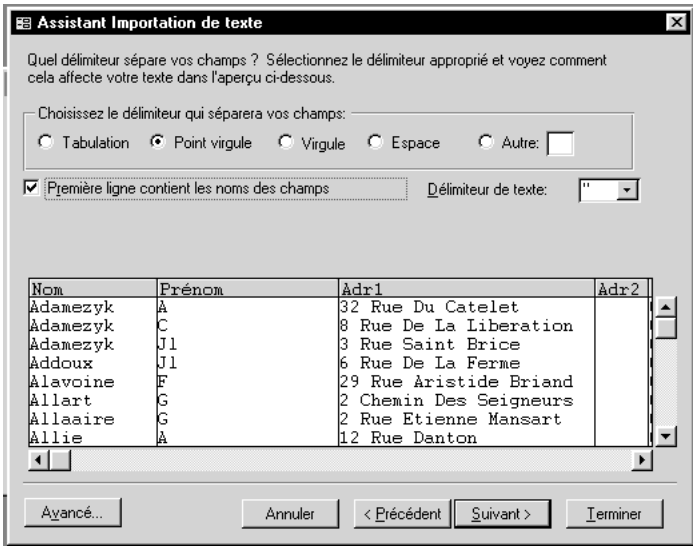
- Charger le fichier **Cours01.mdb**

Importer des données externes

- Menu Fichier / Données externes / Importer
- Dans la boîte Importer / Type de Fichier, choisir Fichiers texte / Sélectionner df-Cli500.txt / Cliquer sur le bouton Importer, Suivant
-



- Dans la boîte Assistant Importation de texte / Cliquer sur le bouton Suivant / Laisser le choix du délimiteur de séparation de champ sur Point-virgule / Cocher le rectangle devant «Première ligne contient les noms des champs»
- Cliquer sur le bouton Suivant / Laisser le choix sur «Dans une nouvelle table» / Cliquer sur le bouton Terminer / Une boîte de dialogue vous annonce : « Importation du fichier dfCli500.txt vers la table DfCli500 terminée» / OK



Mise à jour de la taille des champs

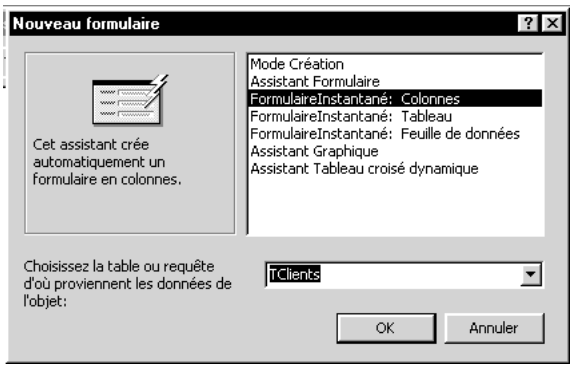
- Onglet Tables / Sélectionner Table DfCli500 / Edition / Renommer / TClients
- Onglet Tables / Sélectionner TClients / Bouton Modifier
- Pour chaque champ, modifier les tailles, enregistrer les modifications

Nom	Texte	(30 caractères)
Prénom	Texte	(30 caractères)
Adr1	Texte	(35 caractères)
Adr2	Texte	(35 caractères)
CodePostal	Texte	(5 caractères)
BureauDis	Texte	(35 caractères)
Néle	Date/Heure	(Date, abrégé)
Profession	Texte	(35 caractères)
Catégorie	Texte	(1 caractère)
PouvoirAchat	Texte	(1 caractère)
Civilité	Numérique	(octet)

L'importation a attribué des valeurs par défaut (Texte 255). Attention, ne pas prendre une taille trop petite.

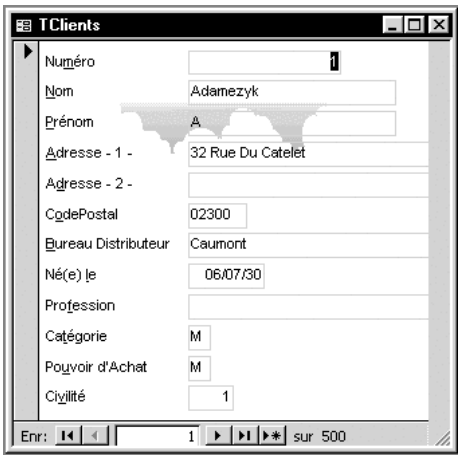
Réalisation d'un formulaire avec l'assistant

- Onglet Formulaires / Nouveau / Formulaire instantané : Colonnes / Choisir la table TClients / Bouton Ok



Création de touches de raccourcis

- Ouvrir le formulaire en mode création en passant par le menu ou par la barre des icônes
- Sélectionner chaque étiquette / changer les légendes. Pour qu'une lettre soit soulignée et devienne ainsi une touche de raccourci, il suffit de mettre le caractère & devant cette lettre.

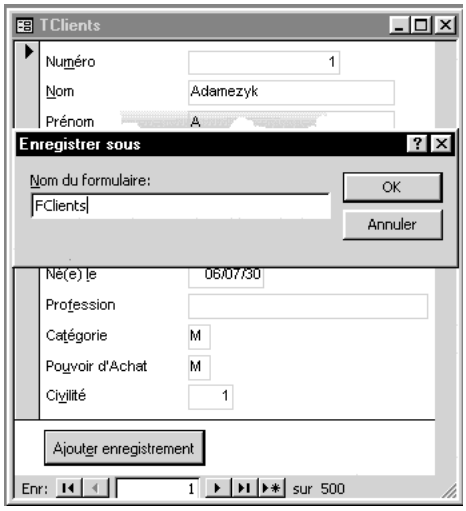


- Exemple, pour l'étiquette Nom, tapez en légende &Nom
- Tester vos raccourcis. Attention, ne pas attribuer le même raccourci à plusieurs objets.

Création d'un bouton avec l'assistant

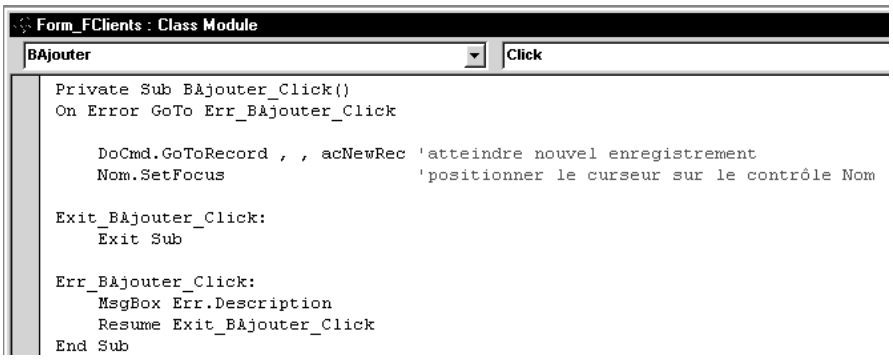
- Formulaire en mode création / Agrandir la partie pied de page du formulaire en plaçant le curseur sous le mot pied de page / Tirer vers le bas
- Cliquer sur l'icône assistant (baguette magique)

- Cliquer sur icône bouton de commande / Se positionner sur la partie pied de page du formulaire / Dessiner un rectangle en gardant le bouton gauche enfoncé / Relâcher la souris
- Assistant Bouton de commande / Catégories : Opérations sur enreg. / Actions : Ajouter un enregistrement / Bouton Suivant
- Sélectionner Texte / Ajout&er enregistrement / Bouton Suivant
- Nommer le bouton BAjouter / Cliquer sur Terminer
- Passer en mode Formulaire / Tester le bouton
- Enregistrer votre formulaire sous le nom FClients



Premier regard sur le code

- Passer en mode création / Sélectionner le bouton Ajouter enregistrement / Ouvrir le menu contextuel en cliquant sur le bouton droit de la souris / Choisir l'option Créer code événement...
- Voici le code généré par l'assistant / Les mots réservés sont en bleu



Commentaires sur le code

- Le code est généré sur l'événement Click, donc le nom de la procédure est composé du nom de l'objet (BAjouter) suivi du caractère souligné (_) et du nom de l'événement (Click).
- DoCmd peut se traduire par Exécuter
- GoToRecord peut se traduire par Aller à l'enregistrement
- acNewRec est une constante de VB pour spécifier Nouvel Enregistrement
- la syntaxe exacte peut être obtenue en sélectionnant le mot GoToRecord, puis en cliquant sur la touche de fonction F1

Syntaxe de la méthode :

```
DoCmd.GoToRecord      [typeobjet, nomobjet]
                      [, enregistrement] [, référence]
```

- Nous pouvons commenter le code. Ici on peut ajouter la mention atteindre nouvel enregistrement en commençant par le caractère ' qui permettra au compilateur d'interpréter les mots qui suivent comme des commentaires et non comme des instructions. Les commentaires apparaîtront en vert.
- Ajoutons l'instruction Nom.SetFocus qui oblige le curseur a se positionner sur le champ (contrôle) Nom. Menu Débogueur / Compiler et enregistrer tous les modules / Fermer / Ainsi, le click sur le bouton Ajouter enregistrement affiche un formulaire vierge, le curseur est sur la zone Nom. L'utilisateur est prêt pour saisir les informations

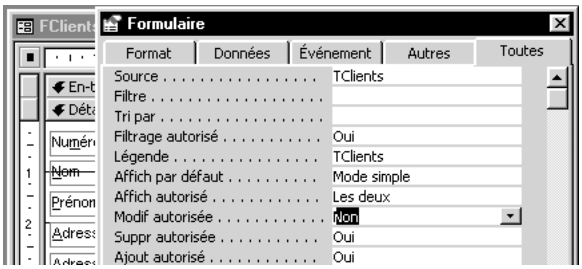
Donc le code généré peut être complété, modifié etc...

- Tapez en lettre minuscule. Les «mots» reconnus par VB prendront automatiquement une majuscule
- Exemples «do» devient «Do, «cmd» devient «Cmd» (abréviation de commande), «gotorecord» devient «GotoRecord
- Les constantes prédéfinies par VB commencent toutes par 2 lettres minuscules, exemple acNewRec
- Regarder dans l'aide (Rubrique d'aide / Onglet Rechercher / 1 Saisissez «const» / 2 Sélectionnez «Constantes» / 3 Cliquez
- «Constantes d'action» / bouton Afficher
- Le code peut être rattaché à tout objet graphique (formulaire, contrôle, étiquette etc. ...).

Nous pouvons maintenant modifier des enregistrements, en ajouter des nouveaux. Mais un problème peut se produire si l'utilisateur modifie une rubrique (zone, champ, contrôle) par mégarde et qu'il passe à un autre enregistrement. Sa modification est sauvegardée automatiquement. Pour éviter cela, nous allons rendre le formulaire en lecture seule.

Protéger les enregistrements déjà créés

- Ouvrir le formulaire en mode création / Double cliquez à l'intersection des 2 règles (rectangle noir) pour afficher les propriétés du formulaire
- Cliquer sur Modif autorisée / sélectionner Non comme valeur pour cette propriété

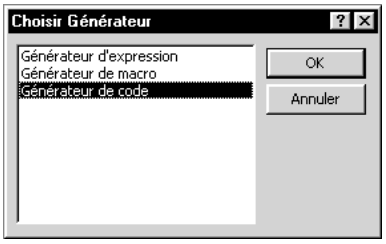


- Fermer la feuille des propriétés
- Activer votre formulaire / Essayer de modifier le prénom (ou un autre contrôle), cela est impossible.

Maintenant que le formulaire est verrouillé, il faut permettre à l'utilisateur de le déverrouiller quand il veut modifier un enregistrement. Donc nous allons concevoir un bouton pour autoriser les modifications. L'assistant ne permet pas de créer ce genre de bouton. Donc nous devons travailler seul.

Créer un bouton de commande sans assistant

- Sélectionner le formulaire FClients / Passer en Mode création / Afficher la boîte à outils
- Désactiver le bouton Assistant en cliquant dessus / Cliquer sur l'outil Bouton de commande
- Dessiner un bouton près du bouton Ajouter enregistrement / Le bouton apparaît immédiatement
- / Il porte le nom Commande 25
- Cliquez sur le bouton avec le bouton droit de la souris pour faire apparaître le menu contextuel / Sélectionner Propriétés
- Changer le nom Commande25 en BEditer, la légende Commande 25 en &Editer / Fermer la feuille de propriétés
- Cliquer sur Editer avec le bouton droit / Choisir Créer code événement / Choisir Générateur / Générateur de code / Bouton OK



- Taper l'instruction Me.AllowEdits=True / Touche Entrée / Tapez Nom.SetFocus / Touche Entrée

> *Me est un nom réservé qui désigne le formulaire actif. Donc ici, on affecte la valeur True à la propriété AllowEdits du formulaire FClients. (AllowEdits signifiant Modification Autorisée). L'instruction SetFocus place le curseur sur le contrôle Nom qui devient actif. N'oublions pas les commentaires. Cela facilite la lecture (et la mémorisation) du code.*

Nous pouvons tester notre bouton.

- Mode formulaire / Essayer de modifier le prénom, cela est impossible /
- Cliquez sur le bouton Editer / Modifier le prénom

Créer un bouton pour sauvegarder les enregistrements

Pour faciliter le travail de l'utilisateur, nous allons créer un bouton Sauvegarder. Utiliser l'assistant. Essayez de le faire seul donc sans lire les lignes d'instructions qui suivent.

- Formulaire mode modification / boîte à outils / Baguette magique / icône bouton de commande / Opérations sur enreg. / Sauvegarder un enregistrement / Bouton suivant / Texte &Sauvegarder / BSauvegarder / Bouton Terminer / Tester

  L'enregistrement est bien sauvegardé même si rien de visuel ne se passe. Nous améliorerons cela ultérieurement.

Créer des procédures évènementielles pour des événements de formulaire

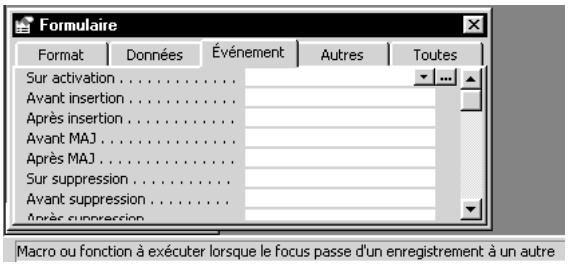
Actuellement, nous avons écrit du code dans des procédures «Click». Il existe de nombreux événements différents. Les événements peuvent se produire sur tout type d'objet (formulaire, contrôle, bouton etc...).

L'art de la programmation événementielle réside dans la faculté d'imaginer les événements auxquels il faut rattacher du code Nous avons protégé notre formulaire contre les modifications accidentelles, mais que se passe-t-il

1. quand on clique sur éditer ? Le formulaire devient modifiable et il le reste. Il serait possible de modifier le code du bouton Sauvegarder pour verrouiller le formulaire après la sauvegarde.
2. si l'utilisateur sauvegarde l'enregistrement sans utiliser notre bouton (par l'intermédiaire du menu) ? Notre formulaire ne serait pas verrouillé.

Dans le premier cas, il faut trouver un événement qui a lieu à chaque fois qu'Access affiche un nouvel enregistrement dans le formulaire. Donc regardons les événements se rapportant au formulaire.

- Formulaire / modifier
- Afficher les propriétés du formulaire / onglet événements
- Le curseur dans la zone Sur activation
- Lire le message en bas de l'écran / Macro ou fonction à exécuter lorsque le focus passe d'un enregistrement à un autre



- Ouvrir la liste déroulante / [Procédure événementielle] / Cliquer sur le bouton avec 3 petits points / La procédure Private Sub Form_Current() s'affiche
- Taper l'instruction Me.AllowEdits=false et ajouter le commentaire 'Modification non autorisée'
- Copier cette ligne dans le presse papier (sélectionner la ligne, CTRL+C)
- Ouvrir la liste déroulante de droite / choisir AfterUpdate / Copier le contenu du presse papier sur la première ligne (sous Private Sub Form_AfterUpdate) en utilisant CTRL+V

Dans cette procédure, qui se déroule quand un enregistrement est modifié, nous ajouterons un message pour l'utilisateur. Ce message placé à cet endroit aura l'avantage d'être exécuté quelque soit la méthode choisie par l'utilisateur pour sauvegarder l'enregistrement (bouton Sauvegarder ou l'option Sauvegarder enregistrement du menu Enregistrements)

- Taper l'instruction MsgBox «sauvegarde effectuée» / MsgBox désigne une boîte message

Ce type de boîte est appelé «boîte de dialogue». Remarquez qu'il est impossible de cliquer sur un autre objet de l'application en dehors du bouton de cette boîte. On dit que la fenêtre est «modale». Cette propriété est très intéressante lorsque l'on désire que l'utilisateur fasse un choix, donne une information complémentaire.

TD2

Sélectionner des enregistrements

Objectifs

1. **Savoir créer une zone de liste modifiable**
2. **Savoir appréhender l'objet RecordSet**
3. **Savoir utiliser le générateur de requête**
4. **Savoir synchroniser un formulaire et une liste**

Début

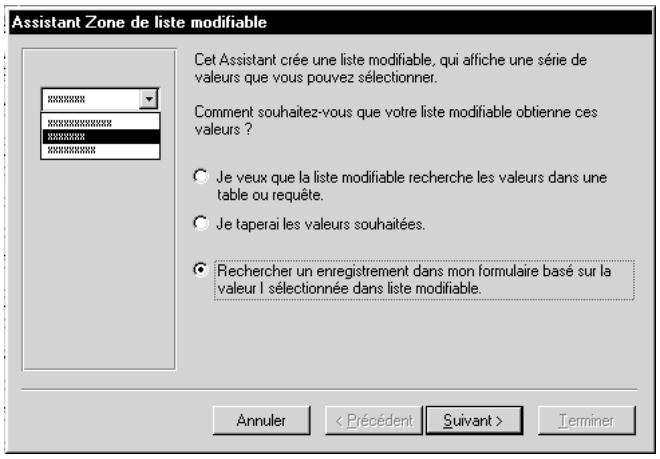
- Charger le fichier Cours02.mdb

  Dans cette base, le formulaire FClients a pour légende Clients. Les boutons créés précédemment ont été supprimés ainsi que le code qui leur était associé. La propriété Modification autorisée du formulaire a été remise à Oui. Le fond a été changé, l'identifiant a pour légende clé.

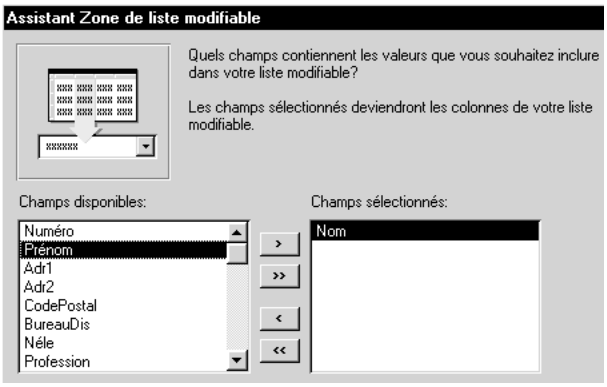
Création d'un zone de liste modifiable

80% du temps passé sur les SGBDR consiste à interroger la base. Donc il est important d'offrir à l'utilisateur des outils efficaces pour effectuer des recherches. L'outil zone de liste modifiable est tout désigné pour remplir ce rôle.

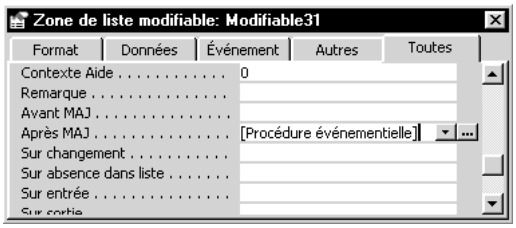
- Ouvrir le formulaire FClients en modification / Augmenter la largeur du formulaire
- Dans la boîte à outils, sélectionner l'assistant (baguette magique)
- Sélectionner l'outil Zone de liste modifiable / Déposer le sur le formulaire
- Assistant Zone de liste modifiable / Cocher «Rechercher un enregistrement dans mon formulaire basé sur la valeur sélectionnée dans liste modifiable » / Bouton Suivant



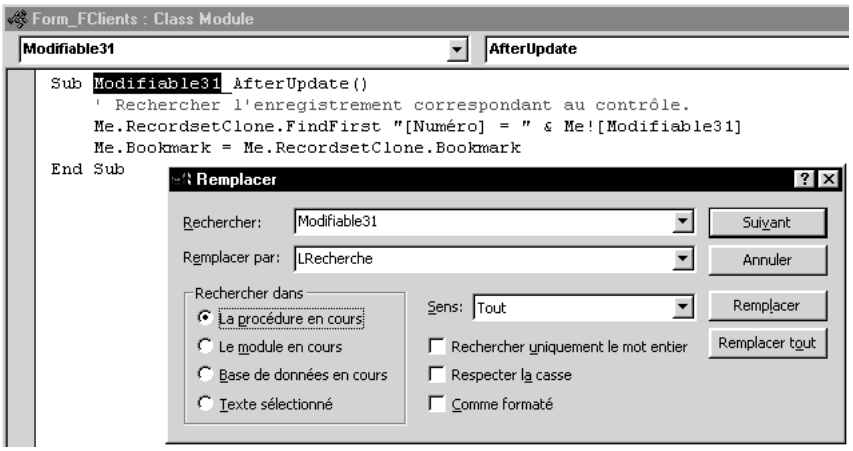
- Double-click sur le champ Nom



- Bouton suivant / Laisser cocher «Colonne clé cachée» / Bouton suivant
- Nommer l'étiquette &Recherche / Bouton Terminer
- Mettre l'étiquette Recherche en caractère gras / Peindre la couleur du fond de la liste en gris clair en sélectionnant la liste et en choisissant le gris (icône pot de peinture couleur fond)
- Dans la feuille de propriété, choisissez Après MAJ / cliquez sur le bouton (...) à côté de [Procédure événementielle].



- Sélectionner Modifiable31 (ou un autre numéro éventuellement) / Menu Edition / Remplacer Modifiable31 par LRecherche dans la procédure en cours / Remplacer tout / Fermer la boîte de dialogue



- Nommer la liste Lrecherche

> *Si vous regardez à Après MAJ, vous constatez que la case est vide. Cliquez sur la flèche, sélectionner [Procédure événementielle].*
> *Puis cliquez sur le bouton 3 petits points pour faire réapparaître le code associé. Donc attention, le changement de Nom d'un objet ne modifie pas l'intitulé des procédures qui lui son rattachées et inversement.*

Découvrir l'objet RecordSet

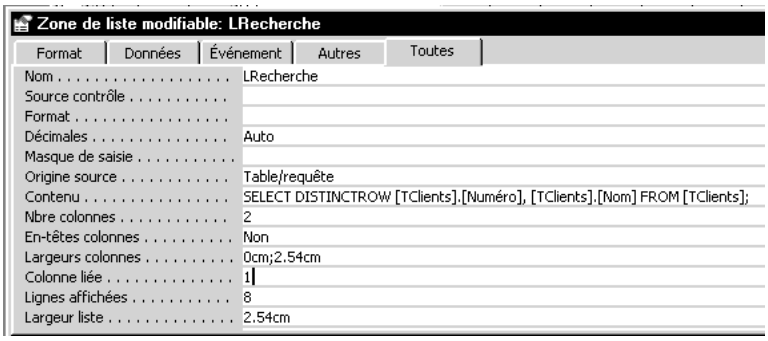
Examinons attentivement le code généré.

```
Sub LRecherche_AfterUpdate()
' Rechercher l'enregistrement correspondant au contrôle.
     Me.RecordsetClone.FindFirst "[Numéro] = " & Me![LRecherche]
     Me.Bookmark = Me.RecordsetClone.Bookmark
End Sub
```

Un commentaire précise le rôle du code. La première instruction peut se traduire par "Trouver le premier enregistrement pour lequel l'attribut numéro est égal à la valeur contenue dans la liste LRecherche". «Me» désigne le formulaire actif «RecordsetClone» désigne le jeu d'enregistrements attaché à ce formulaire. Ici c'est le contenu de la table clients. «FindFirst» signifie «Trouver le Premier». «[Numéro]=» correspond au nom d'un attribut de la table. Ici, il s'agit de l'attribut qui est clé primaire. «Me![LRecherche] désigne le contrôle LRecherche du formulaire actif, ici c'est notre liste modifiable.

La seconde instruction fait appel à un BookMark (signet) qui est un point de repère (pointeur) sur un enregistrement. Donc on demande que la valeur de ce point de repère pour le formulaire soit la même que celle de l'enregistrement trouvé.

Il y a une chose importante qu'il faut comprendre. Dans la liste l'utilisateur voit des noms. Mais l'objet LRecherche manipule 2 attributs (le numéro et le nom). Le numéro est caché car la largeur de sa colonne a été réglée sur 0 cm. Pour vérifier cela, afficher les propriétés de la zone de liste. La ligne «Largeurs de colonnes» affiche «0cm;2.54cm». Le nombre de colonnes est donc bien de 2. La colonne liée est la première, cela signifie que la valeur renvoyée par le contrôle correspond bien à «numéro»



Nous allons vérifier cela en affichant dans un message la valeur du contrôle.

- Tapez l'instruction «MsgBox "Me![LRecherche] = " & Me![LRecherche]» entre les deux instructions de la procédure.
- Afficher le formulaire en mode «Formulaire» / Sélectionner
- «Addoux» dans la liste déroulante
- Le message affiche le numéro 4 / Cliquez sur OK

Le message disparaît. L'instruction suivante s'exécute, le bookmark du formulaire prend la même valeur que celui de l'enregistrement trouvé.

Donc, Attention, l'information affichée peut être différente de l'information manipulée Supprimer le message soit en effaçant la ligne, soit en la transformant en commentaire. Donc, nous disposons maintenant d'un outil de recherche. Nous pouvons le tester. Cela semble bien fonctionner. L'ordre dans la liste est cohérent (ordre alphabétique) mais cela est du au fait que les valeurs existantes ont été importées alors qu'elles étaient triées.

- Vérifions cela en créant un nouveau client (Adonis / Kévin / Place Foch / 02000 / Laon / 01/01/1930 / C / S / 1)
- Fermer le formulaire / Ouvrons le de nouveau / Cherchons dans la liste Adonis / il est à la fin de la liste Cela n'est pas pratique, mais il suffit de préciser dans la requête (attachée au formulaire) que l'affichage doit être trié. Nous en profiterons pour demander l'affichage du prénom.

Utiliser le générateur de requêtes

- Mettre le formulaire en mode Création / Afficher les propriétés de la zone de liste / Cliquez sur la ligne Contenu et sur le bouton (...)
- Dans le générateur de requête / Champ Nom / Ligne Tri / Choisir Croissant
- Remplacer «Nom» dans champ par «[Nom] & ' ' & [Prénom]»
- Cliquez sur icône Exécuter symbolisé par un point d'exclamation (!)

- Le client «Adonis Kévin» apparaît en 5 ème ligne / Fermer le générateur de requête en répondant Oui au message vous demandant de confirmer les modifications apportées.

	Numéro	Expr1
▶	1	Adamezyk A
	2	Adamezyk C
	3	Adamezyk J
	4	Addoux Jl
	501	Adonis Kévin
	5	Alavoine F

- Tester la liste / Tout semble parfait ... presque.

Sélectionner Adonis dans la liste, le formulaire se met sur la fiche d'Adonis. Cliquez sur les icônes de navigation pour revenir sur le premier enregistrement. Le nom affiché dans la liste est resté sur Adonis Kévin

Synchronisation du formulaire avec la liste

Question :

dans quel événement faut-il écrire du code pour assurer une synchronisation entre le formulaire et la liste ?

Réponse :

Form_Current

- Formulaire en mode Création / Cliquez sur l'icône Afficher le code ou choisir Code Exe dans le menu Affichage

- Sélectionner l'objet Form dans la liste de gauche / Sélectionner la procédure Current dans la liste de droite
- Taper le code suivant : [LRecherche] = [Numéro] .Ce qui signifie : Assigner la valeur stockée dans le contrôle Numéro au contrôle LRecherche
- Enregistrer / Tester

Voilà c'est correct.

TD3

Répondre à des événements et contrôler les entrées de données

Objectifs

1. **Savoir répondre à des actions de déplacement dans un formulaire**
2. **Savoir vérifier la validité des informations saisies**
3. **Savoir poser une question et agir en fonction de la réponse**

Début

Charger le fichier Cours03.mdb

Comprendre les formulaires et contrôler les évènements

Chaque action de l'utilisateur vis à vis de l'application (appuyer sur une touche, utiliser la souris, saisir des données) se traduit par un événement.
Pour programmer des événements, vous devez imaginer ceux auxquels il faut répondre et trouver les réponses. Dans le tableau ci-dessous, nous trouvons des actions avec les événements associés

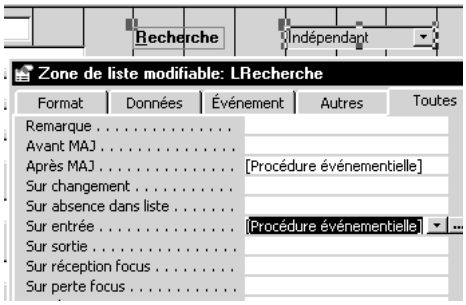
Liste des principales actions et les procédures associées

- Actions utilisateurs Procédures événementielles associées
- Ouvrir et fermer le formulaire Open, Close, Load, Unload
- Accéder à ou sortir d'un formulaire Activate, Deactivate
- Se déplacer de contrôle en contrôle Enter, Exit, GotFocus, LostFocus
- Se déplacer d'enregistrement en enregistrement Current
- Appuyer sur des touches KeyDown, KeyUp, KeyPress
- Cliquer avec la souris Click, DoubleClick, MouseDown, MouseUp
- Modifier des données et sauvegarder des enregistrements
- BeforeUpdate, AfterUpdate
- Ajouter des enregistrements BeforeInsert, AfterInsert
- Effacer des enregistrements Delete, BeforeDelConfirm, AfterDelConfirm
- Filtrer des enregistrements Filter, ApplyFilter

Modifier le couleur de fond d'un contrôle quand il est actif

Nous désirons que la zone de liste modifiable ait une couleur de fond blanc quand elle devient active. Donc quand l'utilisateur «rentre» dans la liste, la couleur doit changer. Nous utiliserons donc l'événement Enter.

- Formulaire Clients en mode modification / Afficher les propriétés de la liste modifiable
- Sélectionner la ligne Sur Entrée / Procédure événementielle / Cliquer sur le bouton (...).
-

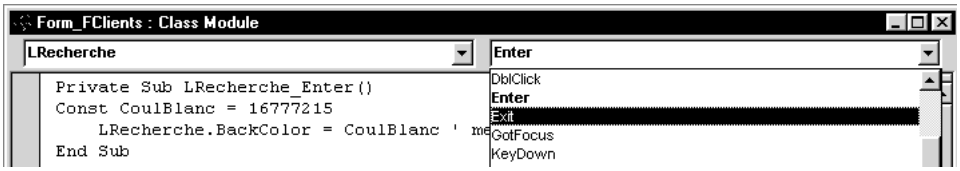


- Taper l'instruction suivante :

```
Const CoulBlanc = 16777215
LRecherche.BackColor = CoulBlanc 'met le fond en blanc
```

Pour que la liste retrouve sa couleur après, il faut écrire le code dans l'événement Exit du contrôle Lrecherche

- Sélectionner l'événement dans la liste déroulante de droite



- Taper l'instruction suivante :

```
Const CoulGris = 12632256
LRecherche.BackColor = CoulGris 'met le fond en gris
```

- Tester

Valider des données

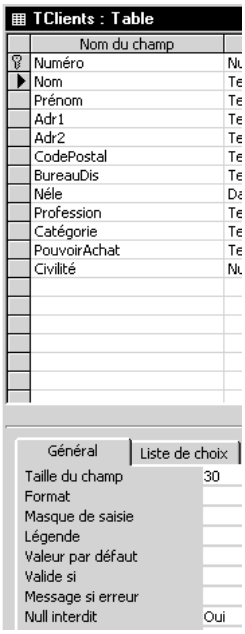
Un SGBDR manipule beaucoup d'informations, les risques d'erreurs sont nombreux (problème de saisie), les conséquences peuvent être importantes (perte de client, de contrat etc. pour une simple erreur d'adresse par exemple). Il vous appartient donc de vérifier et de valider les données saisies. Il existe plusieurs fonctionnalités pour s'assurer de la validité des données :

- le type des données
- la propriété Masque de saisie

- la propriété Null interdit
- la propriété Valide Si
- la procédure événementielle BeforeUpdate

Obliger l'utilisateur à saisir un nom et un prénom

- Ouvrir la table TClients en modification
- Sélectionner l'attribut Nom
- Indiquer oui sur la ligne Null Interdit



- Appliquer le même traitement à l'attribut Prénom

Testons le bon fonctionnement de ces règles de validation

- Ouvrir le formulaire Clients
- Cliquer sur le bouton Nouvel Enregistrement
- Taper Charles dans le contrôle Prénom
- Cliquer sur Sauvegarder enregistrement dans le menu Enregistrements
- Access vous affiche un message d'erreur

- Appuyer sur ESC pour annuler

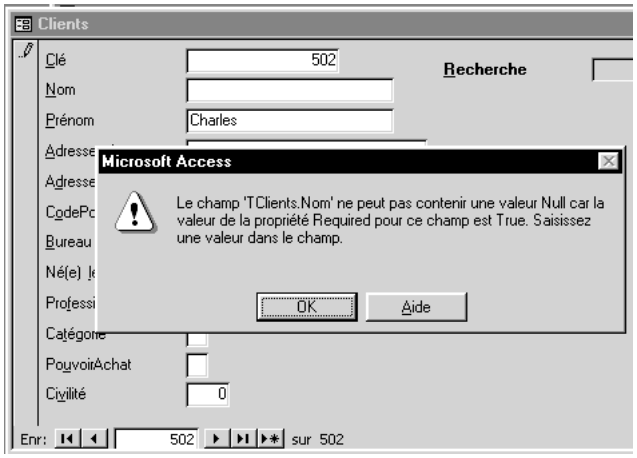
Créer une procédure événementielle vérifiant de code postal

Quand l'utilisateur tente de sauvegarder un enregistrement, l'événement BeforeUpdate se produit. Donc c'est dans cet événement que l'on doit autoriser la sauvegarde si tout est correct sinon il faut «annuler l'événement»

- Ouvrir le formulaire en mode création
- Afficher les propriétés du formulaires
- Cliquer sur AvantMAJ / [Procédure événementielle] / Bouton (...)
- Taper l'instruction suivante

```
'vérification du code postal
If IsNull(CodePostal) Then
     MsgBox " Vous devez indiquer un code postal ", vbExclamation
     CodePostal.SetFocus
     Cancel = True
End If
```

- Tester en supprimant le code postal du client 1
- Menu Enregistrement / Sauvegarder / Votre message apparaît



- Taper sur Esc.

Avons-nous fini ? Non

Première remarque :

Nous pouvions obliger l'utilisateur à saisir le code postal en utilisant la propriété Null interdit pour l'attribut CodePostal dans la création de la table (même démarche que pour le Nom et le Prénom).

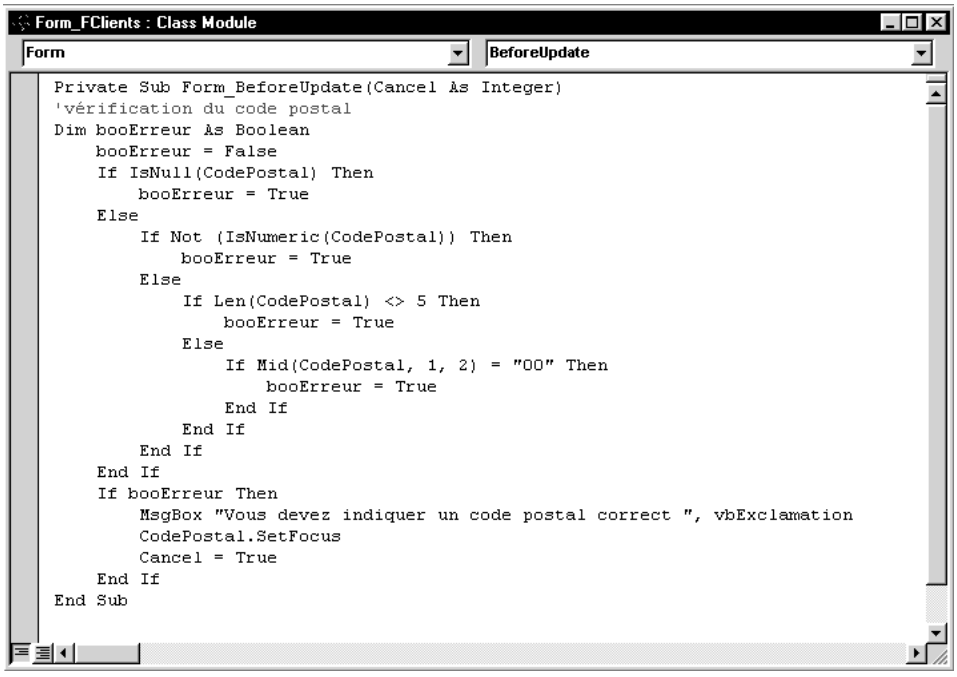
Deuxième remarque :

L'utilisateur doit saisir un code postal, mais est-ce suffisant ? Il peut taper AAAA ou 124. Donc nous devons modifier notre procédure de contrôle. A quelles conditions un code postal est-il bon ? S'il ne contient que des chiffres. S'il y en a 5, pas un de plus ni un de moins. De plus les 2 chiffres de gauche ne doivent pas être «00».

En algo, nous pourrions écrire

```
Si Null(CodePostal) Alors
      Erreur <— vrai
Sinon
      Si Non(Numeric(CodePostal)) Alors
            Erreur <— vrai
      Sinon
            Si Longueur(CodePostal) <> 5 Alors
                  Erreur <- vrai
            Sinon
                  Si SousChaine(CodePostal,1,2) = «00» Alors
                        Erreur <— vrai
                  FinSi
            FinSi
      FinSi
FinSi
Si Erreur Alors
      Afficher «Code Postal incorrect»
      Curseur sur CodePostal
      Annuler sauvegarde
FinSi
```

Ceci se traduit en basic par le code suivant.



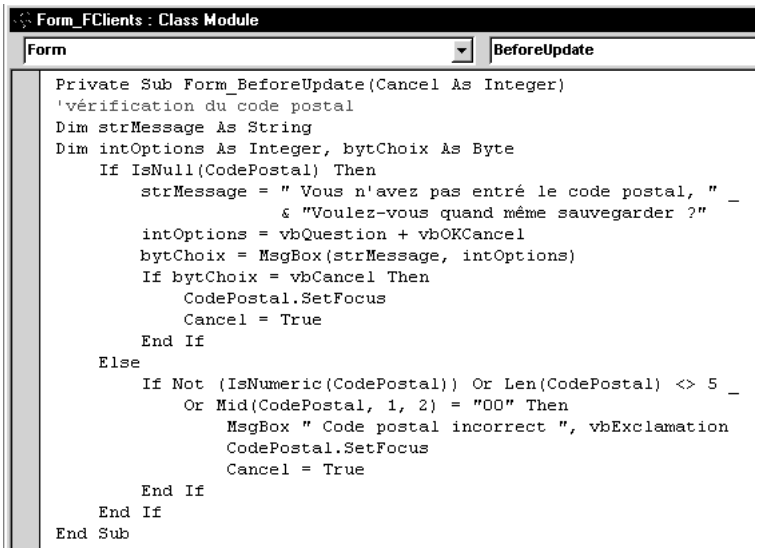
Tester votre procédure en tapant les valeurs suivantes dans le code postal du client 1 : «aaaaa», «00300», «1234». Pensez à choisir Sauvegarder Enregistrement dans le Menu Enregistrement sinon le message ne s'affichera pas (autre possibilité de sauvegarde, se placer sur l'enregistrement suivant)

Il peut être envisageable d'adapter le message au type d'erreur commis par l'utilisateur. Il est aussi possible de regrouper les tests dans la condition Si en utilisant l'opérateur Ou. Voilà une procédure de contrôle efficace, mais vous devez prendre garde à une chose : l'utilisateur ne doit pas être prisonnier de votre application. Je m'explique, actuellement l'utilisateur est obligé de saisir un code postal. Que se passe-t-il s' il ne le connaît pas ?Il reste bloqué et peste contre ces ordinateurs stupides ou (pire encore) il mémorise un code postal «bidon».

Poser une question à l'utilisateur

Donc, nous allons modifier notre procédure.Si le code postal est Null alors nous demanderons à l'utilisateur s'il désire néanmoins effectuer la sauvegarde. Selon son choix, nous l'obligerons à entrer le code postal ou nous sauvegarderons l'enregistrement avec un code postal Null.

Modifions le code de la procédure comme sur l'image suivante.



Nous avons discerné 2 cas, :

- le code postal est Null
- le code postal n'est pas Null.

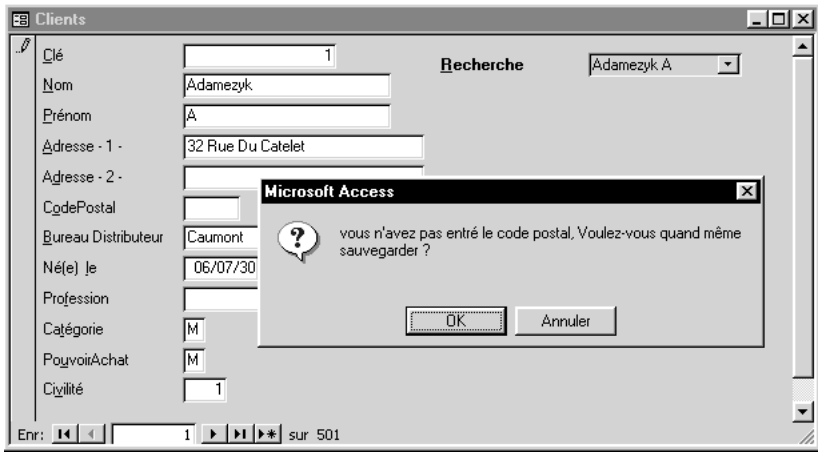
Dans le premier cas la variable strMessage contient le texte à afficher.

Noter que pour écrire une instruction très grande, il est possible de l'écrire sur plusieurs lignes en terminant les lignes par le caractère espace « « suivi du caractère souligné (_) la variable intOptions reçoit vbQuestion et vbOKCancel. Ceux sont deux constantes prédéfinis de VB. Elles sont donc utilisables à tout moment. vbQuestion entraîne l'affichage de l'icône point d'interrogation, vbOKCancel entraîne l'affichage du bouton OK et du bouton Annuler.

MsgBox est une fonction qui passe donc le texte en paramètre ainsi qu'une valeur numérique. La fonction retourne une valeur (selon que l'utilisateur clique sur l'un ou l'autre des boutons). Cette valeur retournée est stockée dans la variable bytChoix.

Si bytChoix contient la valeur vbCancel (l'utilisateur a cliqué sur Annuler) alors nous positionnons le curseur sur le contrôle CodePostal et nous annulons la mise à jour.

Dans le second cas : nous contrôlons la validité du code saisi. Le principe est le même que précédemment mais tous les tests ont été regroupés avec l'opérateur OR. Nous pouvons voir le fonctionnement sur l'image suivante. Pour connaître les valeurs possibles des constantes, sélectionner vbQuestion, puis appuyer sur la touche F1 pour afficher l'aide.



TD4

Ecrire des fonctions personnalisées

Objectifs

1. **Savoir créer un module standard**
2. **Savoir écrire des procédures générales Sub et des fonctions**
3. **Savoir utiliser le passage de paramètres**
4. **Savoir utiliser des procédures générales**

Début

- Charger le fichier Cours04.mdb

Comprendre les modules et les procédures

Pour l'instant, nous avons écrit tout notre code dans les procédures événementielles associées à des objets. Les procédures générales (que nous allons écrire) ne s'exécutent pas automatiquement en réponse aux événements. Il faut dire quand elles devront s'exécuter.

Il y a deux types de procédures générales : Sub et Function alors que toutes les procédures événementielles sont de type Sub.

Pourquoi créer des procédures générales ?

Il est possible d'utiliser des procédures générales pour :

- Exécuter des opérations complexes qui ne rentrent pas dans une expression
- Réutiliser du code pour éviter de répéter une tâche. Exemple, : l'affichage de message. Plutôt que de réécrire les mêmes instructions dans chaque procédure événementielle, il est préférable d'écrire une procédure et de l'utiliser dans chaque procédure événementielle qui nécessite un message.
- Diviser des tâches de programmation en unités plus facilement gérables.

Modules standard et modules de formulaires

Il est possible de mettre des procédures générales dans un module de formulaire (ou d'état), Elles sont alors accessibles à toutes les procédures événementielles du formulaire (ou de l'état). Créez ce type de

procédure pour exécuter une tâche qui s'applique directement à un formulaire, par exemple, pour travailler avec des informations dans les champs du formulaire.

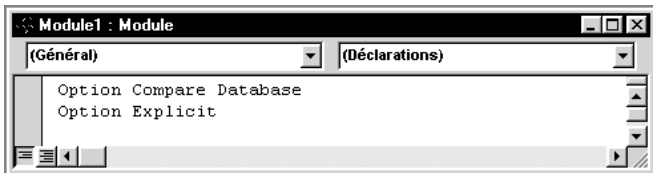
Il est possible de mettre des procédures générales dans un module standard. Elles sont alors accessibles à toute l'application. Créer ce type de procédure pour exécuter une tâche qui s'applique à plusieurs formulaires ou états.

Créer des procédures générales dans un module standard

Une application peut contenir plusieurs modules et chacun d'eux peut contenir plusieurs procédures. Cela permet de créer des modules spécialisés réutilisables dans de multiples applications (il suffit d'importer le module)

Créer un module standard

- Fenêtre base de données / Onglet Modules /Bouton Nouveau
- La fenêtre Module1 : Module s'affiche

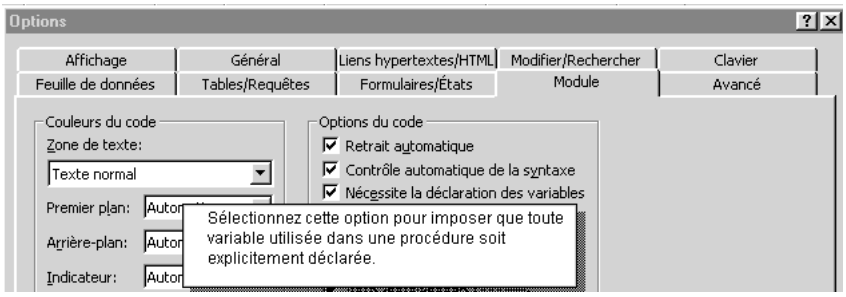


«Option Compare Database» ne peut être utilisée qu'au sein de Microsoft Access. Elle génère des comparaisons de chaînes fondées sur l'ordre de tri déterminé par l'ID des paramètres nationaux de la base de données dans laquelle a lieu la comparaison

Avec l'instruction Option Explicit, toutes les variables doivent être déclarées explicitement à l'aide d'une instruction Dim, Private, Public, ReDim ou Static. Si vous tentez d'utiliser un nom de variable non déclarée, une erreur se produit lors de la compilation.

Important : indiquer toujours Option Explicit

- Menu Outils / Option... / Onglet module/ Activer Nécessite la déclaration des variables.



A l'ouverture, un module est vide ou presque. Il contient une section spéciale appelée Déclarations. Toutes les variables et constantes déclarées dans cette section seront connues du module entier. Le code écrit dans un module est connu de tous les modules de l'application.

Déclarer des valeurs constantes

Déclarons une constante pour le nom de notre application. Tapez sous la ligne Option Explicit l'instruction suivante :

```
Const conNomApp = «Gestion des clients»
```

Cette instruction rend accessible la constante conNomApp à tout votre module. Si vous souhaitez qu'elle soit connue de votre application entière, il faut ajouter le mot Public (anciennement Global) devant le mot Const.

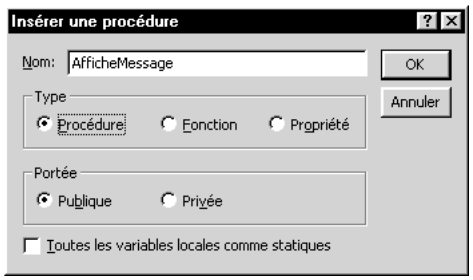
Créer une procédure Sub

Si pour les procédures événementielles liées aux objets de votre application les noms et les paramètres vous sont imposées, là dans votre module, c'est vous qui devez décider du nom et des paramètres éventuellement nécessaires.

Créons la procédure AfficheMessage qui prendra en charge l'affichage des messages que nous adresserons aux utilisateurs

- Cliquez sur le bouton «Insérer une procédure» de la barre d'outils

- Nom «AfficheMessage» / Type Procédure
- Laisser Etendue sur Publique / Bouton OK



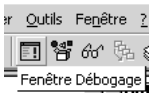
Tapez l'instruction suivante :

```
MsgBox «Voici mon message.» , vbExclamation, conNomApp
```

- Cliquez sur le bouton Enregistrer / Nom du module Mmessages

Nous allons tester notre procédure en utilisant la fenêtre de déboguage

- Cliquez sur le bouton «Fenêtre de déboguage»



- Dans la fenêtre de déboguage, tapez le nom de votre procédure «AfficheMessage»
- Votre procédure s'exécute et affiche la boîte de dialogue avec votre message et icône Point d'exclamation. Le titre de la boîte affiche le nom de votre application

- Cliquez sur OK

Ajouter des arguments à une procédure

Notre procédure doit permettre l'affichage d'un message adapté à une situation. Donc il nous faut un paramètre qui contiendra le texte

- Fermer la fenêtre de déboguage
- Entrez strMessage As String dans les parenthèses de l'instruction Sub
- Modifier la ligne MsgBox en remplaçant la chaîne «Voici mon message» par le nom de la variable qui contiendra le texte à afficher. Cette variable est strMessage
- Tester avec la fenêtre de déboguage / Taper le nom de la procédure suivi du message que vous désirez afficher.

  Exemple : AfficheMessage «Je peux dire tout ce que j'aime»

- Cliquez sur OK

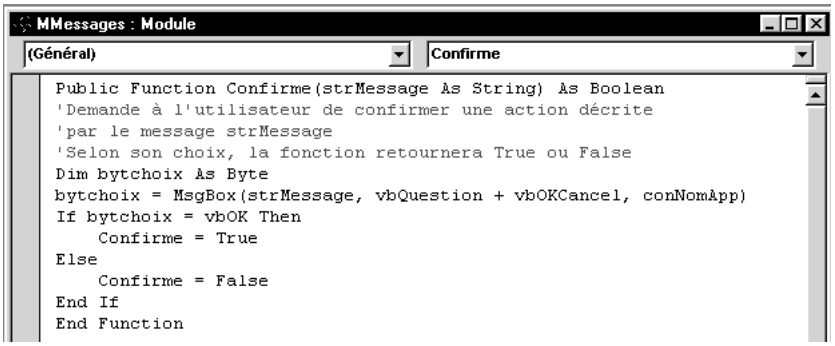
Une fonction permet de retourner une valeur. Dans le code d'une fonction, il doit toujours exister une ligne ayant cette syntaxe :

```
NomDeLaFonction = ValeurARetourner
```

Créer une fonction qui pose une question

Nous allons créer une fonction qui demandera à l'utilisateur s'il veut faire une action ou l'annuler.

- Fermer la fenêtre de déboguage
- Cliquer sur le bouton «Insérer une procédure»
- Nom : Confirme / Type Function / Bouton OK
- Taper le code suivant

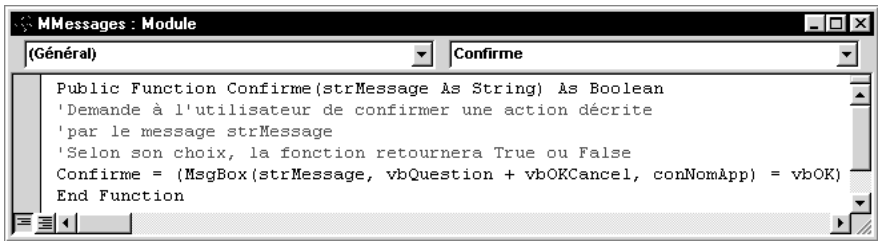


Testons dans la fenêtre de déboguage

- Taper l'instruction suivante : ?Confirme(«Est-ce OK ?»)
- La boîte de dialogue s'affiche / cliquer sur OK / la valeur Vrai s'affiche dans la fenêtre de déboguage

- Si vous recommencez en choisissant le bouton Annuler, la fonction retourne Faux et affiche cette valeur dans la fenêtre de déboguage

Remarque : les instructions de la fonction peuvent être plus compactes.



Utiliser des procédures générales dans un formulaire

Maintenant que nous avons créé des procédures, nous allons les utiliser. Nos procédures étant déclarées publiques sont utilisables dans toute l'application.

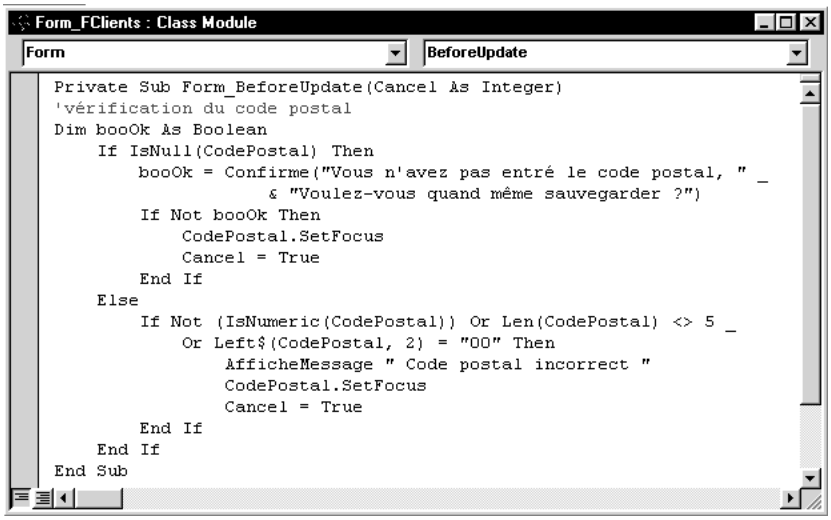
Utiliser la fonction Confirme

- Onglet Formulaire / Sélectionner FClients / Menu Affichage / Code
- Liste de gauche du module : sélectionner l'objet Form / Proc : BeforeUpdate
- Supprimer les variables déclarées
- Créer la variable booOK de type Booléen (dim booOk as boolean)
- Remplacer le code situé entre «If IsNull(CodePostal) Then».et «If bytChoix(vbCancel)» par les lignes suivantes :

```
booOK = Confirme(" Vous n'avez pas entré le code postal," _
          & "Voulez-vous quand même sauvegarder ?")
```

- Remplacer «If bytChoix(vbCancel)» par «If Not booOK»
- Remplacer «MsgBox» dans la ligne «MsgBox «Code postal « par AfficheMessage
- Supprimer «,vbExclamation»

Nous devons avoir le même code que dans l'image suivante.



- Testez le formulaire.

TD5

Améliorer une application

Objectifs

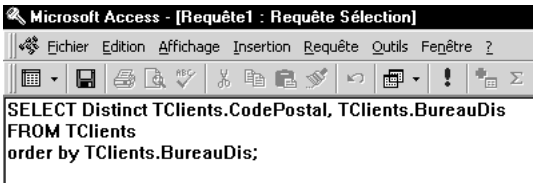
1. **Savoir créer une liste modifiable basée sur une autre table**
2. **Savoir créer et utiliser un formulaire spécialisé**
3. **Savoir activer un formulaire à la demande**
4. **Savoir actualiser le contenu d'une liste modifiable**

Début

- Charger le fichier Cours04.mdb

Notre application comporte encore de nombreux risques d'erreurs de saisie. Pour en avoir la preuve, regardons les différentes valeurs saisies pour le code postal et le bureau distributeur.

- Sélectionner onglet Requêtes / Bouton Nouveau
- Mode création / bouton Ok
- Ajouter une table / TClients / Bouton Ajouter
- Double click sur CodePostal et BureauDis / Affichage SQL
- Ajouter Distinct après Select / Ajouter Order by TClients.BureauDis



- Bouton exécuter (c'est icône avec un point d'exclamation)

Vous pouvez constater des ambiguïtés (Barisis et Barisis Aux Bois; Bertaucourt et Berteaucourt; Brissay Hamegicourt et Brissy Hamegicourt etc...).

CodePostal	BureauDis
02300	Abbecourt
02800	Achery
02700	Amigny Rouy
02800	Andelain
02800	Anguilcourt Le Sart
02270	Assis Sur Serre
02300	Autreville
02700	Barisis
02700	Barisis Aux Bois
02800	Beautor
02800	Bertaucourt
02800	Berteaucourt
02240	Brissay Choigny
02240	Brissay Hamegicourt
02240	Brissy Hamegicourt
02300	Caillouel

- Fermer la base Cours04.mdb et Ouvrir la base Cours05.mdb

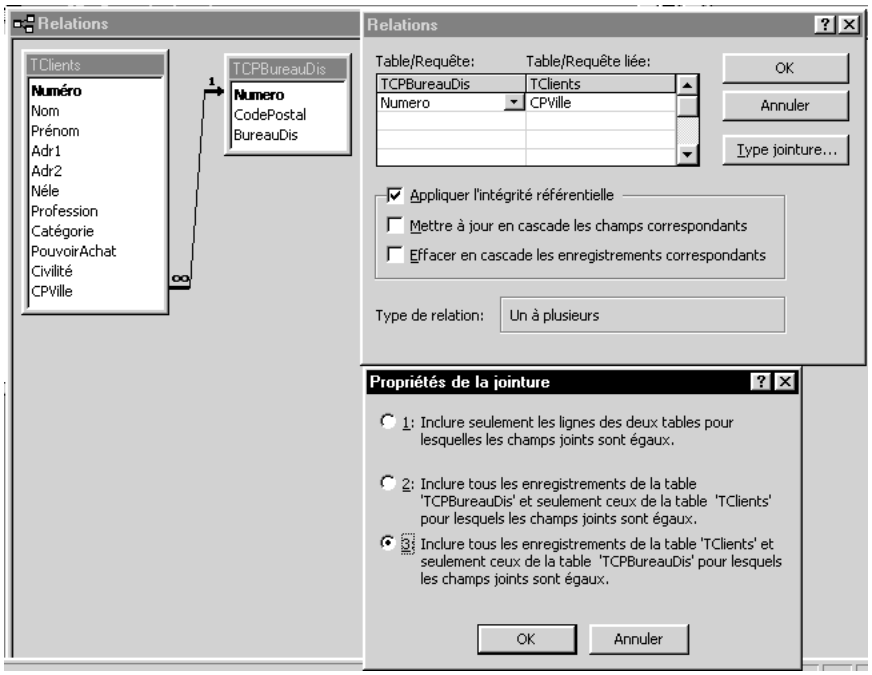
Regardons cette nouvelle base. Elle comporte 4 nouvelles tables :

- TCategorie pour les divers types de catégorie de clients (Jeune, Moyen, Classique)
- TCivilite pour les titres à utiliser dans les courriers (Monsieur, Madame, Mademoiselle)
- TCPBureauDis pour les codes postaux et les villes
- TPouvoirAchat pour les divers types de pouvoir d'achat (Faible, Moyen,
- Supérieur)

La table TClients a été modifiée. Les attributs BureauDis et CodePostal ont été supprimés. Un attribut a été créé : CPVille (c'est une clé étrangère qui sera en rapport avec Numero dans TCPBureauDis)

Etablissement des relations entre les tables

- Menu Outils / Relations / La relation entre TClients et TCPBureauDis existe déjà
- Sélectionner le trait de liaison / Bouton droit de la souris / Modifier
- Dans la boîte de dialogue Relations / Bouton Type jointure
- Déplacer la fenêtre pour voir l'ensemble



Il y a une marque devant Appliquer l'intégrité référentielle. Cela permet d'éviter les incohérences entre les 2 tables.

Propriétés de la jointure : ici c'est l'option 3 qui est choisie. Cela permet une jointure entre la table TClients et TCPBureauDis même si le code postal d'un client n'a pas été défini.

> *Il faut être précis et prudent sur le choix des types de jointure. Surtout ne pas choisir la solution de facilité qui consiste à ne pas appliquer l'intégrité référentielle*

Création des autres relations

- Cliquer sur OK pour fermer les 2 boîtes de dialogue
- Cliquer sur icône Ajouter une table (c'est celui qui a un signe »+» jaune)
- Double click sur les noms de table suivants : TCategorie, TCivilite, TPouvoirAchat
- Bouton Fermer

Pour la relation entre TClients et Tcategorie

- Cliquer sur l'attribut CatCode de TCategorie / Garder le doigt appuyé sur la souris
- Amener le symbole sur Catégorie dans la table TClients
- Appliquer l'intégrité référentielle / Type de jointure 3 / Bouton Ok
- Bouton Créer

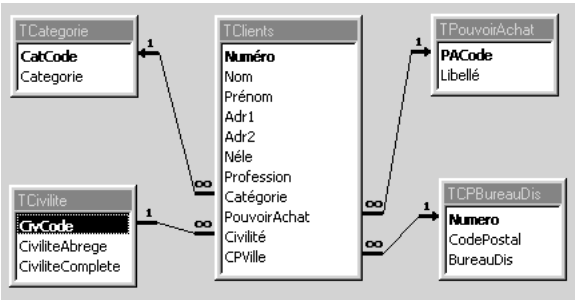
Pour la relation entre TClients et Tcivilite

- Amener l'attribut CivCode sur Civilité de TClients
- Appliquer l'intégrité référentielle / Type de jointure 1 / Bouton Ok

- Bouton Créer

Pour la relation entre TClients et TpouvoirAchat

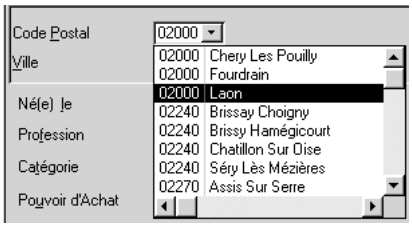
- Amener l'attribut PACode de TPouvoirAchat sur PouvoirAchat dans TClients
- Appliquer l'intégrité référentielle / Type de jointure 3 / bouton Ok
- Bouton Créer



- Fermer la fenêtre Relations et répondre Oui pour enregistrer les modifications

Regard sur le nouveau formulaire

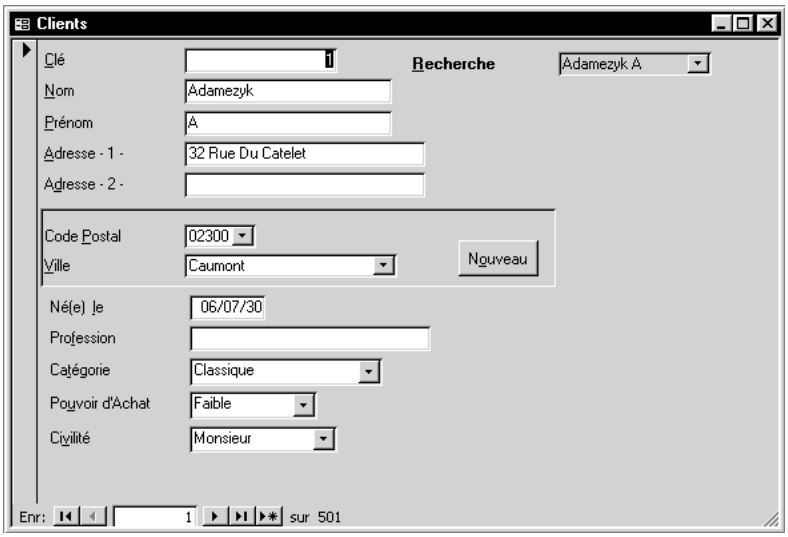
Ouvrir le formulaire Clients en mode Formulaire



Plusieurs listes déroulantes existent (pour le code postal, la ville, la catégorie, le pouvoir d'achat et la civilité). Les listes déroulantes ont été créées en utilisant l'assistant. Chacune d'elles présente les informations existantes dans la base pour faciliter les saisies et éviter ainsi les fautes de frappes.

Attardons nous sur les listes Code Postal et Ville. Le client 1 habite Caumont code postal 02300

- Ouvrir la liste Code Postal / Sélectionner 02000 Laon



- La liste Ville affiche Laon automatiquement
- Ouvrir la liste Ville / Sélectionner Caumont
- La liste Code Postal affiche 02300

Etudions les listes Code Postal et Ville dans le mode création de formulaire.
La liste de Code Postal se nomme CPVille, elle est associée à l'attribut CPVille de la table TClients. Son contenu est associé à la requête suivante :

```
SELECT DISTINCT  TCPBureauDis.Numero,
                 TCPBureauDis.CodePostal,
                 TCPBureauDis.BureauDis
FROM
      TCPBureauDis
ORDER BY
      TCPBureauDis.CodePostal;
```

Elle a 3 colonnes (logique puisque 3 attributs sur la clause Select). Les largeurs sont définies à 0 cm pour le numéro (attribut clé), 1 cm pour le code postal et 5 cm pour le nom de la ville.

L'ensemble est trié sur le code postal. La colonne liée étant la 1, c'est bien la valeur de la clé du tuple sélectionné qui sera stockée dans CPVille (la clé étrangère)

La seconde liste (Ville) se nomme VilleCP (deux contrôles d'un même formulaire ne peuvent avoir le même nom), elle est aussi associée à l'attribut CPVille de la table TClients. Son contenu est associé à la requête suivante :

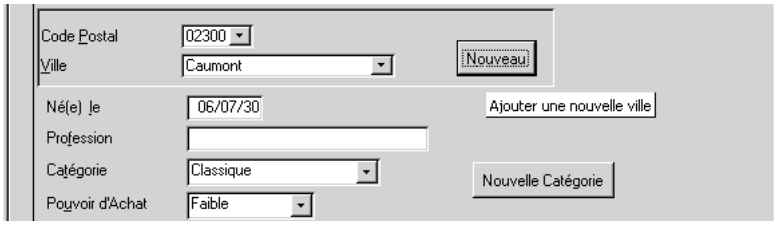
```
SELECT DISTINCT  TCPBureauDis.Numero,
                 TCPBureauDis.BureauDis,
                 TCPBureauDis.CodePostal
FROM
      TCPBureauDis
ORDER BY
      TCPBureauDis.BureauDis;
```

Seul l'ordre des attributs diffère ainsi que l'ordre de tri qui porte ici sur le nom du bureau distributeur.

Ainsi l'utilisateur peut faire la saisie avec l'une ou l'autre de ces listes. Nous constatons donc qu'un même formulaire peut contenir plusieurs contrôles ayant la même source.

Utilisation d'un formulaire spécialisé activable à la demande

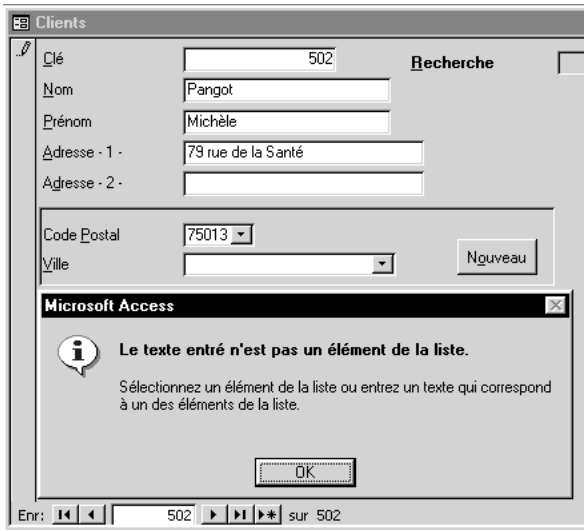
- Ouvrir FClients en mode formulaire
- Créer le client suivant : (Pangot; Michèle; 79 rue de la santé; 75013; Paris
- Access nous retourne un message indiquant que notre choix est limité à la liste



Lorsque nous voulons saisir une valeur qui n'existe pas dans la liste déroulante, nous sommes bloqués. Aussi, un bouton Nouveau est situé à droite. Il doit permettre de saisir une nouvelle ville. Quand la souris se positionne dessus, il serait souhaitable d'afficher une info-bulle

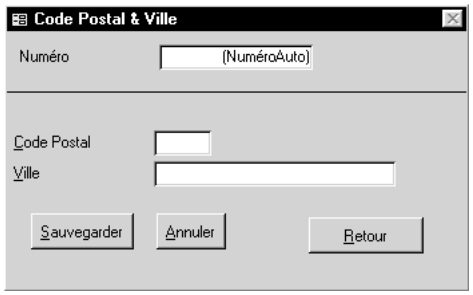
- Passer en mode modification du formulaire
- Sélectionner le bouton Nouveau / Afficher ses propriétés / Onglet Autres
- Texte d'info-bulle / Taper «Ajouter une nouvelle ville»
- Passer en mode formulaire
- Positionner la souris / l'info-bulle s'affiche

Remarque : l'info-bulle ne s'affiche que quand le bouton a le focus, ce qui est plutôt bizarre !!!



Testons son fonctionnement, essayons de recréer le client Pangot

- Taper les informations suivantes : Pangot,Michèle, 79 rue de la Santé, 75013
- Le message s'affiche / cliquer sur Ok
- Cliquer sur ESC pour annuler 75013 dans le contrôle CodePostal
- Cliquer sur Nouveau
- Le formulaire Code Postal & Ville apparaît au milieu de l'écran



- Taper «75013» dans le contrôle Code Postal
- Taper «Paris» dans le contrôle Ville
- Cliquer sur le bouton Sauvegarder
- Cliquer sur le bouton Retour
- Sélectionner 75013, Ville affiche automatiquement Paris

Remarque : le code exe associé au formulaire FCPBureauDis est fortement commenté. Vous pourrez l'examiner tranquillement

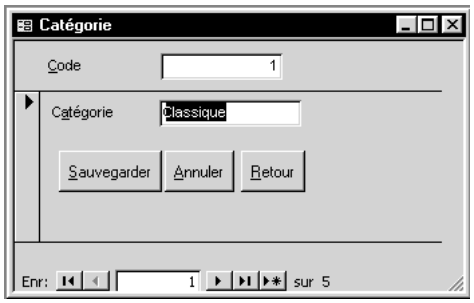
Continuons la saisie.

- Née le 03/09/48
- Profession Secrétaire de Direction
- Catégorie Classique / Pouvoir d'achat Moyen / Civilité Madame
- Sauvegarder avec l'option du menu Enregistrement
- Fermer le formulaire Fclients

Création d'un formulaire spécialisé pour Catégorie

Nous allons créer un formulaire qui permettra l'ajout d'une catégorie de manière interactive

- Créer un formulaire instantané Colonnes basé sur la table TCategorie / Supprimer l'image / Peindre le fond en gris
- Déplacer le contrôle CatCode dans Entête-de-Formulaire / Fond blanc pour les contrôles / Aspect 3D enfoncé
- Ajouter des raccourcis
- Enregistrer le formulaire sous le nom FCategorie
- Légende Catégorie
- Avec l'assistant bouton de commande, créer les boutons Sauvegarder, Annuler et Retour / Nommer les boutons BSauvegarder, BAnnuler et Bretour



- Fermer le formulaire FCategorie
- Ouvrir le formulaire FClients en modification
- Avec l'assistant, créer un bouton qui ouvrira le formulaire FCategorie en affichant tous les enregistrements / Nommer le BNewCat avec le texte Nouvelle Catégorie
- Ouvrir le code événement attaché au bouton
- Modifier le code pour que le formulaire s'ouvre en mode ajout et dans une fenêtre modale

Remplacer le code :

```
DoCmd.OpenForm stDocName, , , stLinkCriteria
```

par le code

```
DoCmd.OpenForm stDocName, , , , acFormAdd, acDialog
```

et supprimer la variable stLinkCriteria.

Le paramètre acFormAdd est une constante prédéfinie indiquant que le formulaire sera ouvert en mode Ajout. acDialog est aussi une constante prédéfinie, elle signifie que le formulaire s'ouvrira dans une fenêtre modale. Maintenant, testons le formulaire.

- Ouvrir FClients en mode formulaire
- Rester sur le premier Client / cliquer sur le bouton Nouvelle Catégorie

Testons les boutons de FCategorie quand rien n'est saisi

- Cliquer sur le bouton Retour / Cela marche, le formulaire se ferme
- Cliquer sur le bouton Nouvelle Catégorie
- Cliquer sur le bouton Sauvegarder / Cela marche (pas d'erreur signalée)
- Cliquer sur le bouton Annuler.

Access vous signale qu'il n'est pas possible d'annuler. Cela est du au fait que les champs sont Null. Donc modifions le code pour autoriser la commande d'annulation quand Categorie n'est pas Null. Access dispose d'une fonction IsNull.

Nous pouvons donc écrire l'algorithme suivant :

```
Si Categorie<>Null Alors
      Annuler l'enregistrement en cours
FinSi
```

Ce qui se traduit par le code suivant :

```
If Not IsNull(Categorie) Then
      DoCmd.DoMenuItem acFormBar, acEditMenu, acUndo, , acMenuVer70
End If
```

- Fermer la boite message d'Access / Cliquer sur Retour / Fermer Clients /
- Ouvrir FCategorie en modification / Modifier le code de BAnnuler_click / Ajouter le test If Not .. comme ci-dessus
- Enregistrer / Fermer FCategorie / Ouvrir FCategorie en mode Formulaire / Cliquer sur le bouton Ajout
- Cliquer sur Annuler / Cela marche, le message n'apparaît plus

Testons les boutons quand une nouvelle catégorie est saisie.

- Taper «Très classique» dans catégorie. Dès le premier caractère entré, le compteur affiche 4.
- Cliquer sur Sauvegarder / L'enregistrement est sauvegardé
- Cliquer sur Annuler / Access vous demande si vous désirez supprimer définitivement votre enregistrement / Cliquer sur Non
- Cliquer sur le bouton Ajout / Saisir Cool dans catégorie / Cliquer sur Retour
- Rouvrir le formulaire, vous pouvez vérifier que «Cool» a été enregistré

Si l'utilisateur a déjà sauvegardé (avec le bouton Sauvegarder), il n'est pas nécessaire que le bouton Retour refasse la même chose. Donc, nous allons utiliser une variable booléenne boolDejaEnregistre qui sera mis à FAUX au départ (form_current). Le bouton Sauvegarder le positionnera à VRAI. Profitons en d'ailleurs pour avertir l'utilisateur que son enregistrement a bien été effectué. Le bouton Retour fera une sauvegarde si boolDejaEnregistre est à FAUX avant de fermer le formulaire. Cette variable booléenne doit

être connue des événements Form_Current, BSauvegarder_Click, BRetour_Click donc il faut la déclarer dans la partie déclaration du module du formulaire.

- Ouvrir FCategorie en modification (icône équerre dans la barre outil)
- Afficher le code en cliquant sur l'icône Afficher Code (ou menu Affichage / Code exe)
- Déclarer la variable sous Option Explicit dans la partie objet (Général), proc : (déclarations)

```
Dim boolDejaEnregistre As boolean
```

- Initialisation à FAUX dans l'événement Form_Current par

```
boolDejaEnregistre = False
```

- Initialisation à VRAI dans l'événement BSauvegarder_Click sous l'instruction DoCmd

```
boolDejaEnregistre = True
```

- Ajout d'un message

```
AfficheMessage " Enregistrement effectué "
```

Dans BRetour, testons le booléen pour savoir si la sauvegarde est déjà faite. Remarquons, que pour exécuter la sauvegarde, il suffit d'appeler la procédure BSauvegarder_Click. Nous obtenons le code suivant :

```
If boolDejaEnregistre Then
     DoCmd.Close
Else
     BSauvegarder_Click
     DoCmd.Close
End If
```

Testons notre travail

- Ouvrir FCategorie en mode formulaire
- Afficher l'enregistrement 2 / Remplacer Jeune par Adolescent
- Cliquer sur Sauvegarder / le message "Enregistrement effectué" s'affiche
- Cliquer sur Retour / le formulaire se ferme normalement (sans rien afficher)
- Ouvrir FCategorie / Afficher l'enregistrement 2 / Remplacer Adolescent par Jeune
- Cliquer sur Retour / le formulaire se ferme après avoir affiché notre message

Attention, si nous ouvrons le formulaire et que nous cliquons sur Annuler quand catégorie contient quelque chose, le message d'Access nous informe encore que l'on ne peut utiliser Annuler. Pourquoi ? C'est simple, nous ne pouvons annuler que de nouvelles informations (informations ajoutées, informations modifiées). Le problème des informations ajoutées a déjà été résolu. Pour gérer les modifications, utilisons un booléen boolAChange qui sera initialisé à FAUX dans Form_Current. L'événement Categorie_Change le positionnera à VRAI. Dans Annuler_Click, nous transformerons le test :

```
If Not IsNull(Categorie) Then
```

par

```
If Not IsNull(Categorie) And boolAChange Then
```

et il faut réinitialiser le booléen à faux. Donc après le if then et avant le end if, il faut écrire

```
boolAChange=False
```

De même, quand on clique sur Retour à partir d'un enregistrement existant, le message «Enregistrement effectué» s'affiche. Complétons là aussi le code

```
If boolDejaEnregistre or not boolAChange Then
```

Pour BSauvegarder_click, il faut aussi ajouter le test If boolAChange. Il reste à écrire le code qui mettra à jour le contenu de la liste Categorie dans le formulaire FClients.

- Ouvrir FClients en mode modification / Sélectionner la liste Catégorie / Afficher les propriétés
- Nommer la liste (actuellement appelée Modifiable41) LCategorie
- Fermer FClients
- Ouvrir FCategorie / Afficher le code BSauvegarder_Click
- Tapez l'instruction suivante sous l'instruction DocCmd

```
Forms!FClients!LCategorie.Requery
```

Cette instruction peut se traduire par : exécuter la requête (Requery) attachée au contrôle LCategorie situé dans le formulaire FClients (membre de la collection des formulaires (Forms)).

Utilisation des opérateurs ! et .

Les opérateurs ! et . (point) dans un identificateur vous permettent d'indiquer le type d'élément qui vient après.
L'opérateur ! indique que ce qui suit est un élément défini par l'utilisateur.
L'opérateur . (point) indique généralement que ce qui suit est un élément défini par MicroSoft Access. (objet, méthode, propriété)

Attention, cette instruction impose donc que le formulaire FClients soit ouvert en mode Formulaire. Donc, deux possibilités s'offrent à nous :

1. le formulaire FCatégorie ne sera ouvert que par le formulaire FClients
2. le formulaire FCatégorie peut être utilisé en dehors de FClients

Dans le premier cas, il faut tester dans l'événement Form_Open du formulaire FCategorie si le formulaire FClients est ouvert. Si celui-ci n'est pas ouvert alors il faut prévenir l'utilisateur et fermer aussitôt le formulaire Fcategorie

Dans le second cas, il faut effectuer le test quand on veut mettre à jour la liste LCategorie. Nous allons opter ici pour le premier cas. Donc écrire ces instructions dans l'événement

```
Form_Open du formulaire FCategorie :
     If Not EstOuvert("FClients") Then
```

```
            AfficheMessage "Ce formulaire doit être utilisé _
            exclusivement avec Clients"
        DoCmd.Close
End If
```

La fonction EstOuvert figure dans le module MFormulaires. C'est une adaptation de la fonction EstChargé que l'on peut trouver par exemple dans l'application «Comptoir» fournie avec.Access. Soignons la présentation du formulaire Fcategorie

- Ouvrir FCategorie en mode création / Afficher les propriétés
- Barre défilement / Aucune
- Afficher Sélecteur / Non
- Boutons de déplacement / Non
- Boîte contrôle / Non
- Boutons Min Max / Aucun
- Bouton Fermer / Non

.

Testons notre travail.

- Ouvrir FCategorie en mode formulaire / Notre message s'affiche

Voilà, l'ensemble paraît satisfaisant.

TD6

Découvrir le Code Data Access Object

Objectifs

1. **Savoir répondre à l'événement Notinlist**
2. **Savoir Utiliser les onglets Database et RecordSet dans du code**
3. **Savoir utiliser le code DAO**

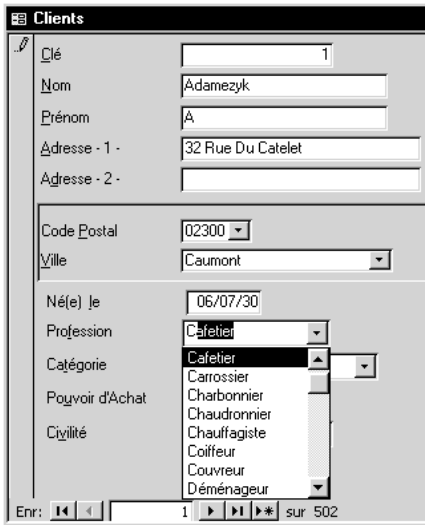
Début

Charger le fichier Cours06.mdb

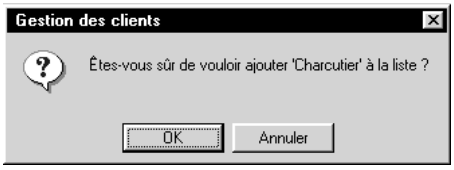
Notre application a été modifiée. Une table TProfession a été créée, la table TClients contient une clé étrangère "Profession" en liaison avec l'attribut ProCode clé primaire dans Tprofession

Tester une zone de liste modifiable « interactive »

- Ouvrir le formulaire FClients en mode formulaire
- Le client 1 exerce la profession de «Charcutier»
- Dans le contrôle Profession, tapez le caractère «C» / «Cafetier» s'affiche immédiatement



- Ouvrir la liste / Elle pointe sur le premier métier dont le libellé commence par «C»
- Compléter votre saisie en «Charcutier» / Valider votre saisie avec la touche ENTREE
- Un message s'affiche, il vous demande de confirmer l'entrée de «Charcutier» dans la liste

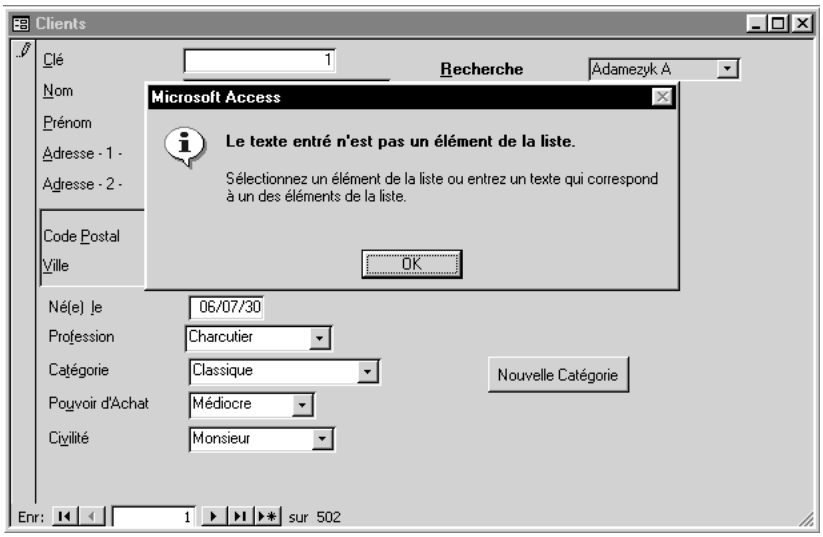


- Répondre OK / Vérifier que la nouvelle valeur figure bien dans la liste
- Passer au client 2, il est «Carriste»
- Taper «Carriste» / Cliquer sur Annuler au message qui s'affiche car Cariste ne prend qu'un seul «r» / Un message vous informe que «Carriste» ne sera pas ajouté.

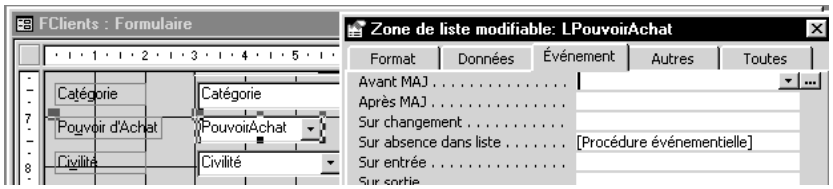
- Corriger l'orthographe et valider

Nous avions vu dans le TD précédent l'utilisation d'un formulaire pour ajouter des informations dans la liste. Là, nous sommes en présence d'une autre technique. Nous allons appliquer ce principe pour la liste Pouvoir d'Achat.

- Ouvrir le formulaire Clients / Taper Médiocre dans la rubrique Pouvoir d'achat pour le client 1
- Le message d'Access s'affiche/ Annuler (Ok / ESC) et fermer le formulaire



- Ouvrir FClients en modification / Sélectionner Pouvoir d'Achat / Afficher les Propriétés
- Sélectionner l'onglet Evénement / Un événement s'appelle «Sur absence dans
- Liste»
- Mettre [Procédure événementielle] / Cliquer sur le bouton «3 points»



La procédure événementielle NotInList dispose de 2 paramètres.

```
Private Sub LPouvoirAchat_
NotInList(NewData As String, Response As Integer)
```

Le premier (NewData) est de type chaîne de caractères (String). Il contient l'information saisie par l'utilisateur. Le second (Response) est de type entier (Integer). Il doit contenir une valeur bien précise. En fonction de la valeur contenue dans Response, Access adaptera son traitement.

Pour connaître les valeurs possibles, nous pouvons nous retourner vers l'aide (Sélectionner NotInList / F1). Nous apprenons alors que les différentes valeurs possibles sont : acDataErrDisplay (Valeur par défaut. Affiche le message par défaut destiné à l'utilisateur. Vous pouvez utiliser cette constante si vous voulez empêcher l'utilisateur d'ajouter de nouvelles valeurs à la liste de la zone de liste modifiable.

acDataErrContinue n'affiche pas de message par défaut pour l'utilisateur. Vous pouvez utiliser cette constante pour afficher un message personnalisé destiné à l'utilisateur. Par exemple, la procédure événementielle peut afficher une boîte de dialogue personnalisée demandant à l'utilisateur s'il veut enregistrer la nouvelle entrée. Si la réponse est Oui, la procédure événementielle ajoute la nouvelle entrée à la liste et affecte acDataErrAdded à l'argument Response. Si la réponse est Non, la procédure événementielle affecte acDataErrContinue à l'argument Response.

acDataErrAdded N'affiche pas de message à l'intention de l'utilisateur, mais vous permet d'ajouter l'entrée dans la liste de la zone de liste modifiable dans la procédure événementielle NotInList. Lorsque l'entrée est ajoutée, Microsoft Access met à jour la liste en actualisant la zone de liste modifiable. Microsoft Access vérifie ensuite à nouveau la chaîne de caractères par rapport à la liste et enregistre la valeur de NewData dans le champ auquel la zone de liste modifiable est liée. Si la chaîne ne figure pas dans la liste, Microsoft Access affiche un message d'erreur.

Maintenant que nous connaissons les paramètres et le mode de fonctionnement, nous devons nous poser la question suivante : que doit faire notre traitement ? Réponse : il doit ajouter une valeur dans la liste.

Attention, cela est une vision simplifiée. Premièrement, il faut demander à l'utilisateur s'il veut vraiment ajouter la valeur. Pour cela nous disposons de notre fonction Confirme. S'il répond Oui alors il faut faire l'ajout sinon nous l'informons que la valeur ne sera pas ajoutée dans la liste en utilisant notre procédure Message.

Maintenant réfléchissons à l'ajout. La liste ne fait qu'afficher le contenu de la table TPouvoirAchat. Donc, ajouter une valeur cela signifie ajouter une valeur dans la table. Pour cela nous utiliserons des objets.

Déclarons une variable pour le message

```
Dim strMessage As String 'pour afficher la question
Déclarons un objet de type DataBase
Dim dbsClients As DataBase 'variable objet de type DataBase
Cet objet nous permettra de travailler sur notre base.
```

```
Déclarons un objet type RecordSet
Dim rstPouvoirAchat As Recordset 'variable objet de type RecordSet
Initialisons notre variable strMessage
strMessage = " Êtes-vous sûr de vouloir ajouter " & "'" _
            & NewData & "'" & " à la liste ? "
```

Remarque : Pour rendre clairement visible la valeur de NewData, nous l'encadrons par des guillemets simples. Attention, les polices de caractères ne permettent pas toujours de bien différencier le guillemet simple du guillemet double quand ils sont écrits l'un à côté de l'autre (exemple : "ajouter'" & NewData & "' à la liste"). Donc, si vous voulez éviter les problèmes, constituer une chaîne ne contenant que le guillemet simple encadré de part et d'autre d'un guillemet double. Assurez vous qu'il n'y ait pas d'espace devant et derrière le guillemet simple.

Commençons notre test par

```
If Confirme(strMessage) Then
```

Puis écrivons le traitement de manipulation de nos objets:

```
Set dbsClients = CurrentDb()
Set rstPouvoirAchat = dbsClients.OpenRecordset("TPouvoirAchat")
rstPouvoirAchat.AddNew
rstPouvoirAchat!Libellé = NewData
rstPouvoirAchat.Update
Response = acDataErrAdded
```

Explications :

Les variables objets doivent être initialisées. Pour cela, il faut utiliser la commande SET. L'initialisation de notre objet dbsClients se fait en utilisant CurrentDb(), c'est à dire la base en cours d'utilisation. L'initialisation de notre objet rstPouvoirAchat (de type RecordSet) se fait avec TPouvoirAchat. Nous pouvons traduire l'instruction par : Ouvrir la table TPouvoirAchat de l'objet DataBase dbsClients, et «stocker» le contenu dans l'objet RecordSet rstPouvoirAchat. Puis nous précisons que nous voulons faire un ajout en utilisant la méthode AddNew. L'instruction suivante affecte NewData à l'attribut Libellé de l'objet rstPouvoirAchat. L'objet rstPouvoirAchat étant «associé» avec la table TPouvoirAchat, il dispose donc de tous les attributs de la table TPouvoirAchat. Enfin nous utilisons la méthode UpDate de l'objet RecordSet pour rendre la mise à jour effective. Comme la mise à jour est opérée, la constante acDataErrAdded est affectée à la variable Response Complétons les instructions pour le cas où l'utilisateur opterait de ne pas ajouter la valeur dans la liste.

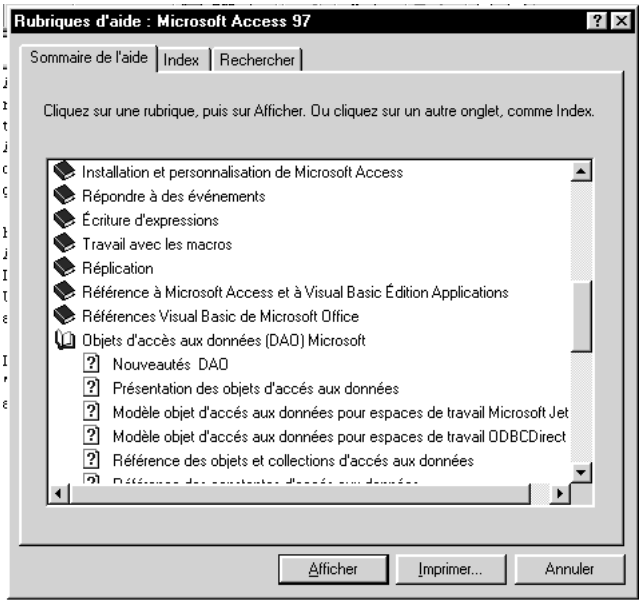
```
Else
     AfficheMessage   " Le pouvoir d'Achat " & "'" & NewData & "'" & _
                 " ne sera pas ajouté dans la liste "
     Response = acDataErrContinue
End If
```

Voilà, ces quelques lignes permettent l'insertion d'une nouvelle valeur dans la Table associée à notre liste. Celle-ci s'enrichit donc d'une nouvelle valeur. Ces lignes sont appelées du code DAO (Data Access Object ou objet d'accès aux données). Il est important de se rappeler que pour manipuler des objets, il faut les déclarer. Puis il faut assigner des objets «réels» aux objets déclarés en utilisant la commande SET.

Après seulement, vous pouvez utiliser les méthodes et les instructions appartenant à la bibliothèque de code DAO.

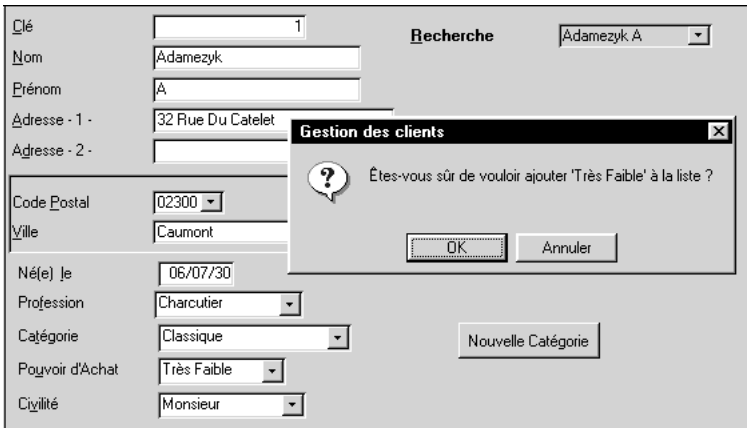
Pour plus d'informations sur le code DAO, consulter l'aide.

- Onglet Sommaire
- Objets d'accès aux données (DAO) ....



Testons notre travail

- Enregistrer / Fermer la fenêtre de code / Fermer le formulaire
- Ouvrir FClients / Tapez «Très Faible» dans le contrôle Pouvoir d'Achat du client 1
- La boîte de dialogue s'affiche et pose la question



- Répondre par OK
- Recommencer la manipulation avec «Très élevé» / Puis choisir le bouton Annuler
- Le message s'affiche, informant l'utilisateur que l'ajout n'est pas effectué

Maintenant que nous avons vu cette méthode, si nous l'appliquons à la liste des civilités, il faut tenir compte du fait que la table TCivilité contient 2 attributs (CiviliteAbrege et CiviliteComplete). La liste dans le formulaire FClients affiche la valeur de CiviliteComplete donc pour CiviliteAbrege, il faut demander à l'utilisateur quelle doit être la valeur à stocker. Pour cela, nous pouvons l'interroger de manière interactive en utilisant la fonction InputBox.

Examinons le code commenté de LCivilite_NotInList. Donc dès l'affectation de NewData avec le code DAO, nous demandons la valeur grâce aux instructions suivantes :

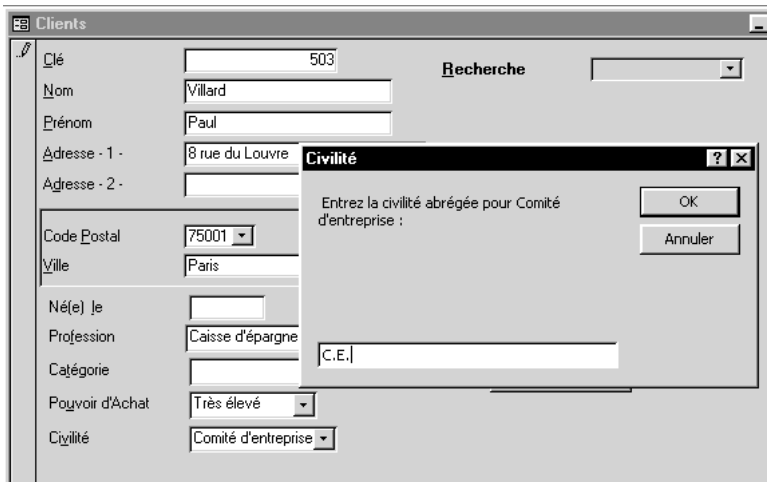
```
strMessage = " Entrez la civilité abrégée pour " & NewData & " : "
strCiviliteAbrege = InputBox(strMessage, "Civilité")
```

La variable strCiviliteAbrege «récupère» la valeur renvoyée par la fonction Inputbox. Puis nous affectons strCiviliteAbrege à l'attribut CiviliteAbrege de l'objet recordset :

```
rstCivilite!CiviliteAbrege = strCiviliteAbrege
```

Pour visualiser le principe, créons un nouveau client. C'est un comité d'entreprise. Nous mémorisons le nom du contact «Villard Paul», l'adresse est «8 rue du Louvre 75001 Paris», la profession «Caisse d'épargne», le pouvoir d'achat est «Très élevé».

Pour civilité, nous rentrerons «Comité d'entreprise», en abrégé «C.E.». Normalement, tout doit se passer correctement. Pour le nouveau code postal, nous pouvons remplacer le message d'Access par notre propre message.



- Afficher la procédure événementielle VilleCP_NotInList
- Taper les instructions suivantes :

```
AfficheMessage " La ville " & "'" & NewData & "'" & _
" ne figure pas dans la liste " & vbCrLf & _
" Utilisez le bouton Nouveau S.V.P. "
SendKeys «{ESC}» 'permet d'annuler la valeur saisie
Response = acDataErrContinue
```

- Sélectionner tout le code / Edition Copier

- Afficher la procédure événementielle CPVille_NotInList
- Edition Coller / remplacer «La ville » par «Le code postal»
- Enregistrer / Tester

Vous pouvez faire la même chose pour la liste des catégories.

TD7

Boîtes de dialogue et états

Objectifs

1. **Savoir filtrer un état à l'aide d'une boîte de dialogue**
2. **Savoir Utiliser les codes d'erreurs**
3. **Savoir écrire du code de gestion des erreurs**

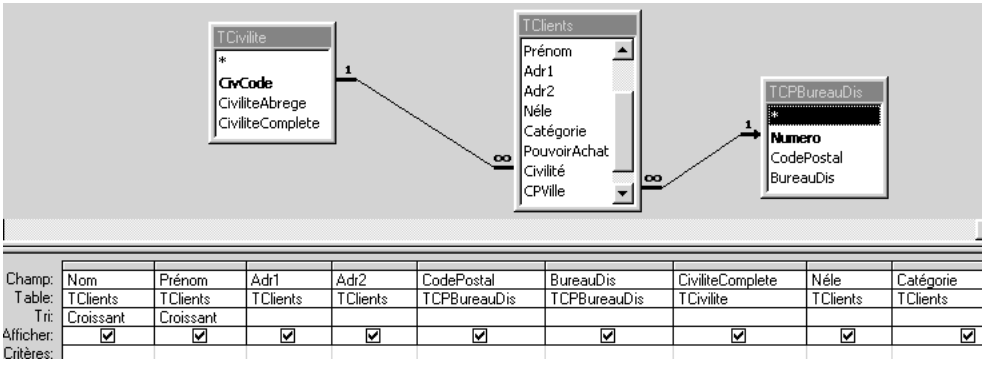
Début

- Charger le fichier Cours07.mdb

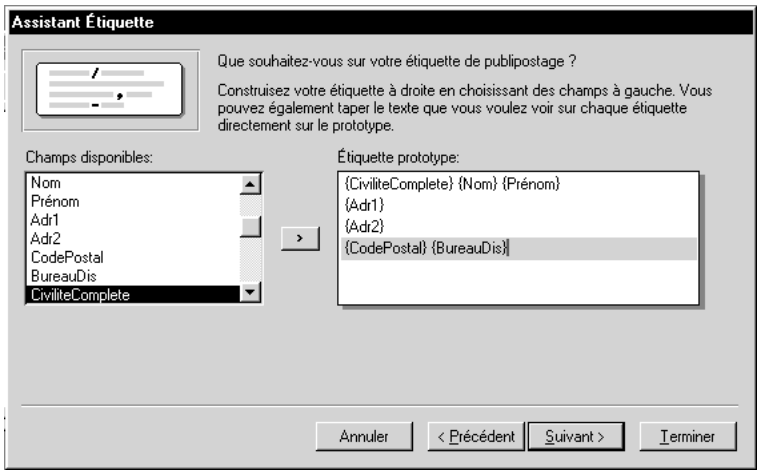
Nous voulons obtenir des étiquettes pour les mailings. Nous envisageons 3 types de mailing :

1. Mailing commun à tous les clients
2. Mailing pour les clients dont l'anniversaire à lieu ce mois
3. Mailing pour les clients d'une catégorie choisie par l'utilisateur

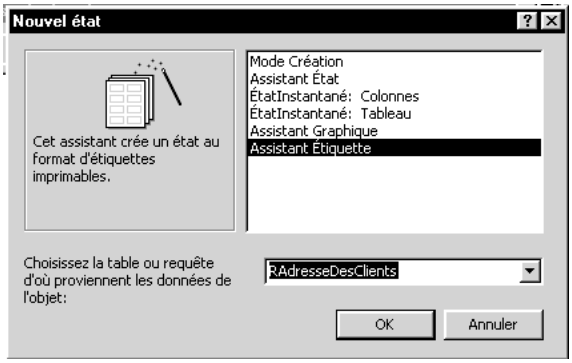
Au lieu de définir 3 états différents, nous utiliserons un seul état et nous filtrerons les enregistrements selon le type de mailing envisagé. Comme les traitements font intervenir la date de naissance, la catégorie, nous avons réalisé une requête RAdresseDesClients basée sur les tables TClients, TCivilite, TCPBureauDis



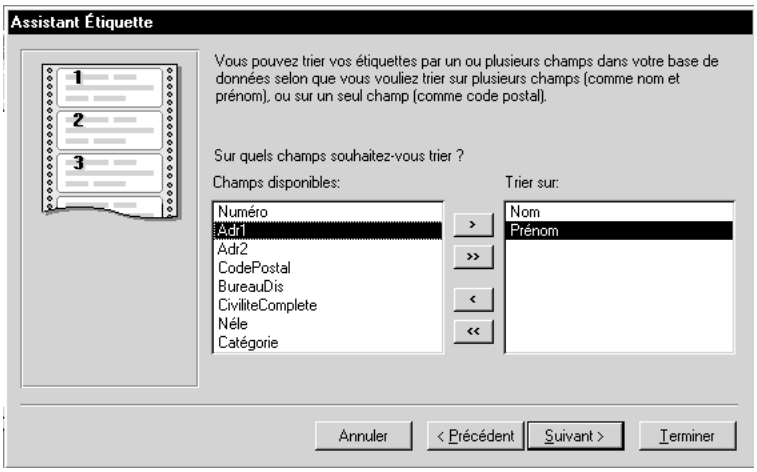
Nous allons réaliser un état EadresseClients en utilisant l'assistant étiquette avec RAdresseDesClients comme source des données.



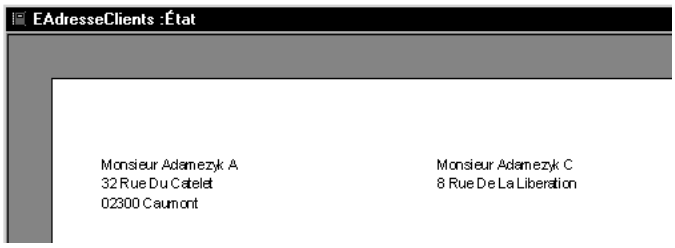
- Etat / Nouveau / Assistant Etiquette / Source RAdresseDesClients / OK / Suivant / Suivant
- Définir le prototype en double-cliquant sur CiviliteComplete / Barre espace
- Double-cliquer sur Nom / Barre espace / Double-cliquez sur Prénom / Entrée
- Double-click sur Adr1 / Entrée
- Double-click sur Adr2 / Entrée
- Double-click sur CodePostal / Barre espace / Double-click sur BureauDis / Bouton Suivant
-



- Trier sur Nom, Prénom / Bouton suivant

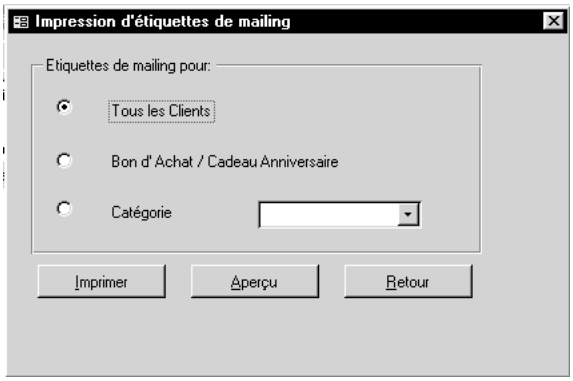


- Nommer l'état EAdresseClients / Bouton Terminer / l'aperçu s'affiche



- Fermer l'état

Pour gagner du temps, le formulaire FEtiquettes existe déjà. Ce formulaire s'ouvre comme une boîte de dialogue. L'utilisateur dispose de trois choix grâce à une boîte à options. Trois boutons sont définis mais seul le code du bouton Retour fonctionne. Nous allons écrire le code pour les 2 autres boutons.



Commençons par le bouton Aperçu

- Ouvrir FEtiquettes en modification
- Afficher le code de BAperçu_Click
- Déclarons une variable strFiltre pour mémoriser le filtre à appliquer sur l'état

```
Dim strFiltre As String
```

Selon le choix de l'utilisateur, il faut définir ce filtre. Le choix est stocké dans MailingType qui est le nom du groupe d'options. Pour le cas Tous les clients, le filtre est inutile donc strFiltre=«»

Pour le cas Bon d'Achat / Cadeau d'anniversaire, nous devons sélectionner les clients dont le mois de naissance est égal au mois de la date système, nous utiliserons la fonction Month qui retourne le numéro du mois de la date passée en paramètre. Le filtre est donc strFiltre = "Month([Néle])=Month(Date())"

Pour le cas de la catégorie sélectionnée dans la liste, il faut que l'attribut catégorie de TClients soit égal à la valeur de la liste LCategorie. Le filtre est donc strFiltre = "[catégorie]= " & Lcategorie

Nous écrirons donc l'instruction Select Case suivante :

```
Select Case MailingType
     Case 1 ' tous les clients
```

```
            strFiltre = «»
      Case 2 ' Clients dont le mois de naissance est celui en cours
            strFiltre = "Month([Néle])=Month(Date())"
      Case 3
            strFiltre = "[catégorie]= " & LCategorie
End Select
```

Puis, nous demandons l'ouverture de l'état en mode aperçu, et en appliquant un filtre sur les données associées à l'état. DoCmd.OpenReport "EAdresseClients", acViewPreview, , strFiltre

Attention, lorsque nous construisons des chaînes «complexes», il est souhaitable de les afficher dans une boîte message pour mieux repérer les erreurs éventuelles. Donc ajoutons l'instruction suivante

```
MsgBox "strFiltre = " & strFiltre & "***"
```

Lorsque tout fonctionnera correctement, nous pourrons supprimer le message. Puis, fermons la boîte de dialogue «FEtiquettes» avec l'instruction suivante

```
DoCmd.Close acForm, "FEtiquettes"
```

Testons le formulaire

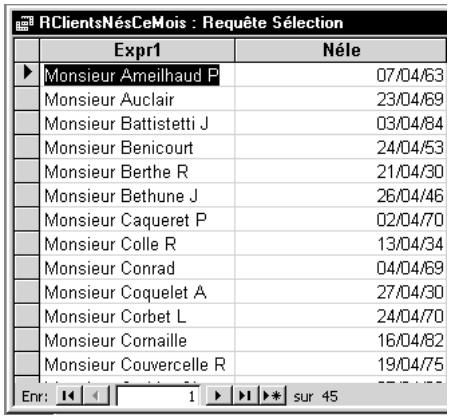
- Fermer la fenêtre de code
- Ouvrir «FEtiquettes» en mode formulaire
- Sélectionner Tous Les Clients / Bouton Aperçu
- L'état s'affiche, la boîte message nous montre que le filtre est vide. Ceci est normal, puisque nous voulons tout le monde

- Cliquons sur Ok, la boîte message se ferme ainsi que le formulaire «FEtiquettes»

Regardons l'état, nous constatons que les clients ayant des adresses incomplètes figurent dans l'état. Ce point sera à revoir. Continuons notre test.

- Fermer l'état / Ouvrir FEtiquettes / Sélectionner Bon d'Achat - Cadeau Anniversaire / Le filtre s'affiche dans la boîte de dialogue

- Bouton Aperçu / (pour contrôle, l'image suivante nous informe que 45 clients sont nés en Avril)



- Fermer l'état / Ouvrir FEtiquettes / Sélectionner la catégorie Classique
- Cliquer sur Catégorie / Bouton Aperçu / 10 clients figurent dans cette catégorie
- Recommençons en choisissant la catégorie «Cool» / bouton Aperçu
- L'état affiche #erreur car la requête n'a pas trouvé de clients correspondant à notre choix

Donc, nous devons supprimer 2 problèmes :

1. les adresses incomplètes
2. les listes vides

Le problème des adresses incomplètes

Il peut être résolu de 2 manières :

1. modifier la requête en réalisant une équijointure entre TClients et TCPBureauDis
2. compléter le test strFiltre pour qu'il évite les codes postaux NULL. Pour cela, déclarons une variable supplémentaire

```
Dim strCPVille As String
strCPVille = "not(isNull([CodePostal])) and not (isNull([BureauDis]))"
```

et modifions le Select de la façon suivante :

```
Select Case MailingType
     Case 1 ’ tous les clients
          strFiltre = strCPVille
     Case 2 ' Clients dont le mois de naissance est celui en cours
          strFiltre = "Month([Néle])=Month(Date())" & " and " _
                    & strCPVille
     Case 3
          strFiltre = "[catégorie]= " & LCategorie & " and " &
     strCPVille
End Select
```

Testons déjà cela avant d’apporter d’autres modifications.

- Ouvrir FEtiquettes en mode formulaire / Sélectionner tous les clients / Bouton Aperçu

Le client Adamézyck C dont l’adresse était incomplète n’apparaît plus

Monsieur Adamezyk A
32 Rue Du Catelet
02300 Caumont

Monsieur Adamezyk J
3 Rue Saint Brice
02300 Caumont

Monsieur Addoux Jl
6 Rue De La Ferme
02800 Versigny

Les listes vides

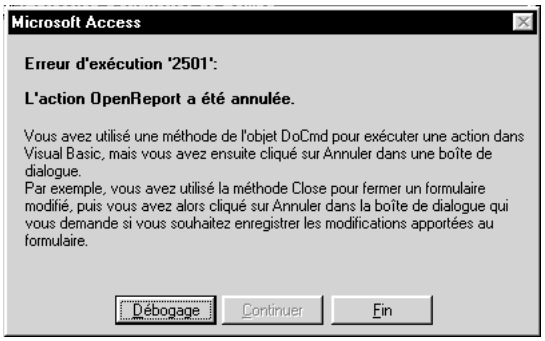
Pour éviter une édition «vide», il existe une méthode NoData qui s’applique aux objets Report. Donc nous allons afficher un message quand la requête ne trouve pas de clients.

- Ouvrir l’état EAdresseClients en modification / Affichage code EXE
- Sélectionner la procédure événementielle Report_NoData
- Taper le code suivant

```
Dim strMsg As String
strMsg = "Aucun client ne correspond à votre choix, _
          l'édition est annulée"
AfficheMessage strMsg
Cancel = True
```

Testons notre travail

- Ouvrir FEtiquettes en mode formulaire / Sélectionner Cool dans la liste des catégories
- Cliquer sur Catégorie / Bouton Aperçu / Le message s’affiche / Bouton Ok
- Access affiche un message «Erreur 2501» / Cliquer sur Fin



L'instruction ouverture de l'état a été annulée par nos soins, donc il faut empêcher l'affichage du message d'erreur. Pour cela, ajouter la commande suivante juste sous la déclaration de la procédure :

```
Private Sub BApercu_Click()
On Error GoTo Err_BApercu_click
```

A la fin de la procédure, juste avant End Sub, ajouter :

```
Exit_BApercu_click:
     Exit Sub
Err_BApercu_click:
     If Err <> 2501 Then
          AfficheMessage   "Numéro erreur : " & Err & _
                           " description : " & Err.Description
     End if
Resume Exit_BApercu_click
```

L'instruction On Error Goto est une instruction fondamentale de la gestion des erreurs imprévues. Elle signifie que si une erreur intervient à tout moment pendant l'exécution d'une instruction de notre procédure, le programme devra se brancher sur l'étiquette Err_BApercu_click. Une étiquette se reconnaît facilement car elle est suivi du caractère (:) (2 points). Dans cette étiquette, on demande d'afficher tous les messages pour les erreurs différentes de l'erreur 2501. Après le Si .. FinSi, l'instruction demande de poursuivre le traitement à l'étiquette Exit_BApercu_click

Testons de nouveau. Cela se passe bien.

Par contre, pour l'utilisateur, quand il effectue un choix dans la liste des catégories, il ne devrait pas avoir besoin de cliquer sur Catégorie. Le rond devant Catégorie retourne la valeur 3. Donc il suffit d'affecter la valeur 3 à mailing type quand l'utilisateur entre dans la liste.

Ajouter les instructions suivantes dans LCategorie_AfterUpdate

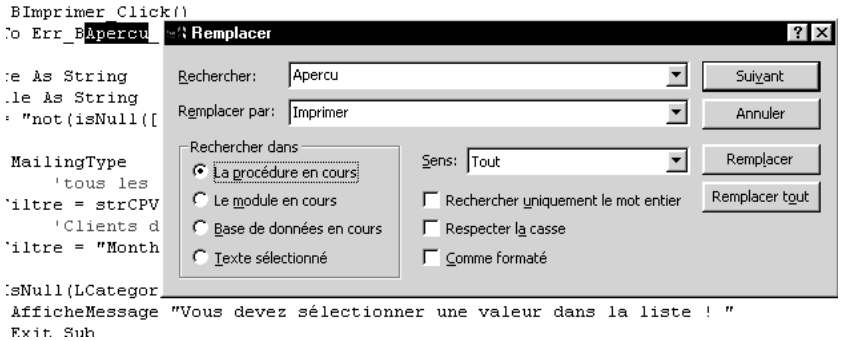
```
Private Sub LCategorie_GotFocus()
     MailingType = 3
End Sub
```

De même, que se passe-t-il si l'utilisateur clique sur catégorie sans choisir de valeur dans la liste ? Réponse : le message "Aucun client ne correspond à votre choix, l'édition est annulée" défini précédemment s'affiche. Mais il est préférable d'afficher un message plus en rapport avec la situation. Ce code devra être exécuté quand l'utilisateur clique sur Aperçu. Donc, ouvrir la procédure événementielle BApercu_Click et ajouter les lignes suivantes :

```
Case 3
If IsNull(LCategorie) Then
AfficheMessage (" vous devez sélectionner une valeur dans la liste ! ")
Exit Sub
Else
strFiltre = "[catégorie]= " & LCategorie & " and " & strCPVille
End If
```

Profitons en pour supprimer l'instruction MSGBox strFiltre située un peu plus bas Voilà, nous avons enfin terminé le traitement du bouton Aperçu. Il reste le bouton Imprimer.

- Sélectionnons tout le code de BApercu_click / Edition Copier
- Ouvrir BImprimer_Click / Edition Coller
- Sélection Aperçu dans la ligne On Error Goto / Edition Remplacer /
- Remplacer par Imprimer / La procédure en cours / Bouton remplacer tout



Le code est donc le même que pour l'aperçu. Mais il faut remplacer la constante acPreview par la constante acNormal

Testons. Tout est OK

Remarque : il serait préférable de créer une procédure pour la partie commune du code

TD8

Réaliser une automation avec VBA

Objectifs

1. **Savoir réaliser une requête complexe**
2. **Savoir utiliser l'explorateur d'objets**
3. **Savoir utiliser On Error Resume Next**
4. **Savoir ajouter un contrôle ActiveX**
5. **Savoir paramétrer les options**

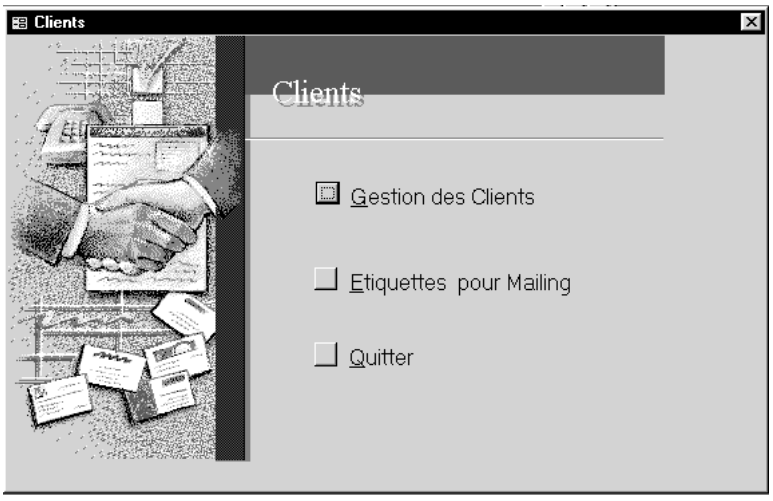
Début

Nous allons découvrir un aspect de l'automation. Attention, l'automation nécessite une machine puissante (processeur / mémoire centrale).

- Charger le fichier Cours08.mdb

Les changements

L'application dispose d'un formulaire menu général FMenuGeneral qui permet d'activer les formulaires (FClients, FEtiquettes), et de quitter l'application. L'application contient un module MConstantes où sont définies des constantes utilisables par toute l'application.



- Global Const conNomApp = " Gestion des clients "
- Global Const conNomAuteur = " Stage VBA - 28 & 29 Avril 1997 "
- Global Const conChemin = "c:\documents\etudiants\stage-vba\"
- Global Const conFicBase = "cours08.mdb"

Dans le formulaire FEtiquettes, la procédure générale CreerFiltre a été définie, elle est appelée dans BApercu_Click et BImprimer_Click. Un bouton pour lancer l'impression des bons d'achat existe, mais aucun code n'est défini.

Fusion avec Word

Nous voulons réaliser un mailing de bon d'achat pour les clients dont c'est l'anniversaire. Ce mailing sera envoyé en début de chaque mois. (une autre règle de gestion peut être définie). La partie gestion des étiquettes a été réalisée dans le cours précédent. Maintenant, il faut réaliser la lettre type. Pour le montant, nous allons utiliser une table TParamètres. Cette table existe déjà, elle contient un attribut BonAchat. La fusion sera basée sur une requête qui permettra d'écrire ce type de «message»

> *Monsieur Duchemin D,*
>
> *bientôt le 17-4, aussi nous sommes heureux de vous offrir ce chèque cadeau d'un montant de 200.00F pour votre anniversaire. Etc ..*

Réalisation de la requête pour les anniversaires

- Passer en mode Création de Requête / Ajouter les tables TCivilite, TClients, TCPBureauDis, TParametres
- Changer la relation entre TClients et TCPBureauDis en double cliquant sur le trait de liaison
- Choisir l'option 1 qui correspond à une équi-jointure
- Afficher le champ CiviliteComplete, modifier le en tapant : Civilite: [CiviliteComplete] & " "
- Afficher le champ Nom, modifier le en tapant : Identite: [Nom] & " " & [Prénom]
- Trier dans l'ordre croissant
- Ajouter [Néle], modifier le en tapant : Month([Néle])
- Sur la ligne «Critères» , taper Month(Date()) pour sélectionner les clients nés le mois courant
- Ne pas afficher ce champ
- Ajouter le champ [Néle] que vous modifiez en JJMM: Jour([Néle]) & "-" & Mois([Néle])
- et enfin, afficher le champ [BonAchat]
- Enregistrer la requête sous le nom RBonAnniversaire
- Enregistrer / quitter Access

Réalisation du document sous Word

- Ouvrir Word
- Taper le texte figurant sur l'image suivante

«Civilite» «Identite»,

Bientôt le **«JJMM»**, aussi nous sommes heureux de vous offrir ce chèque cadeau d'un montant de **«BonAchat»** pour votre anniversaire.

Nous espérons vous rencontrer très prochainement.

A très bientôt …

- Insérer l'image Gateau2.cgm
- Enregistrer sous le nom 97BonAnniv-ori.doc, les mots entre «» sont des champs de fusion
- Quitter Word en enregistrant les modifications.

> *Attention, l'embellissement de la lettre avec un clip art allonge le temps de fusion de manière non négligeable.*

- Relancer Access / Ouvrir cours08.mdb
- Ouvrir FEtiquettes en création / Afficher le code exe
- Afficher le code de l'événement click du bouton BAnniversaire
- Taper l'instruction suivante : FusionWord97 entre les 2 lignes boolBonImprime= ... FusionWord97 est une procédure déjà écrite, elle appartient au module du formulaire puisque spécifique au formulaire.

Avant d'analyser son code, nous allons regarder le fonctionnement de ce code. Pendant le fonctionnement, regardez les messages qui s'afficheront dans les barres d'états ainsi que les boutons actifs de la barre des tâches.

- Ouvrir FEtiquettes en mode exécution
- Cliquer sur le bouton Imprimer les bons Anniversaires

Nous pouvons constater la lenteur du processus. Il y a aussi un autre problème, un second processus application Access est démarré. Ce processus reste actif à la fin du traitement. Avant de tenter de résoudre le problème, étudions le code complet et commenté de cette procédure.

```
Public Sub FusionWord97()
Dim strNomFicModele As String
Dim strNomFicFusion As String
Dim strNomFicMailing As String
Dim strNomFicBase As String
Dim strMessage As String
Dim intchoix As Integer
Dim intAttente As Integer
EtiGauge.Visible = True
CtlGauge.Visible = True
EtiGauge.Caption = " Le traitement est relativement long, soyez patient, Merci "
strNomFicModele = conChemin & "97BonAnniv-ori.doc"
strNomFicFusion = conChemin & "97BonAnniv.doc"
strNomFicMailing = conChemin & "BonAnniv.doc"
strNomFicBase = conChemin & conFicBase
Dim docWord As Word.Document
Dim appWord As Word.Application
```

```
'-------------------------------------------------------------------------------
On Error Resume Next
Kill strNomFicFusion 'suppression des anciens fichiers
Kill strNomFicMailing 'fusion et mailing
'-------------------------------------------------------------------------------
On Error GoTo Err_fusionword97
FileCopy strNomFicModele, strNomFicFusion 'copie du modèle vers le fichier fusion
Set appWord = CreateObject("Word.Application.8") 'initialisation des objets
Set docWord = appWord.Documents.Open(strNomFicFusion) '
EtiGauge.Caption = " Connexion et exécution de la requête .... "
intAttente = DoEvents 'rend la main au système
Stage -VBA - les 28 & 29 Avril 1997 Page 5
Cours 08 - DIDIER Freddy
appWord.Visible = True 'rend Word visible
'------------------------- realisation de la fusion
docWord.MailMerge.OpenDataSource Name:=strNomFicBase, _
LinkToSource:=False, _
AddToRecentFiles:=False, _
Revert:=True, _
Connection:="Query RBonAnniversaire", _
sqlStatement:="SELECT * FROM [RBonAnniversaire]"
EtiGauge.Caption = " Fusion en cours .... "
intAttente = DoEvents 'rend la main au système
With docWord.MailMerge
.Destination = wdSendToNewDocument
With .DataSource
.FirstRecord = wdDefaultFirstRecord
.LastRecord = wdDefaultLastRecord
End With
.Execute pause:=True
End With
EtiGauge.Caption = " Sauvegarde de la fusion .... "
'------------------------- sauvegarde du fichier résultat dans BonAnniv.Doc
' et affectation du fichier à l'objet docWord
appWord.ActiveDocument.SaveAs FileName:=strNomFicMailing, FileFormat:= _
wdFormatDocument, AddToRecentFiles:=True
docWord.Close 'fermeture de 97BonAnniv.doc
Set docWord = appWord.ActiveDocument 'association avec BonAnniv.doc
EtiGauge.Caption = " Traitement du mailing .... "
intAttente = DoEvents 'rend la main au système
'------------------------ traitement du mailing
strMessage = "L'impression peut durer plusieurs minutes." & vbCrLf & _
"Seul le premier bon va être imprimé pour vérifications " & vbCrLf & _
"Voulez-vous lancer l'impression ?"
intchoix = ConFirmeOuiNon(strMessage)
If intchoix = vbYes Then
EtiGauge.Caption = " Impression en cours .... "
docWord.PrintOut Range:=wdPrintCurrentPage ' impression
page courante
strMessage = "L'impression est-elle terminée ? "
intchoix = ConFirmeOuiNon(strMessage)
Else ' quitter
strMessage = "Voulez-vous quitter Word ? "
intchoix = ConFirmeOuiNon(strMessage)
End If
intAttente = DoEvents 'rend la main au système
While intchoix <> vbYes
intchoix = ConFirmeOuiNon(strMessage)
intAttente = DoEvents 'rend la main au système
Wend
EtiGauge.Caption = " Déconnexion en cours .... "
'----------------------- fermeture du document et de l'application Word
' libération de la mémoire
docWord.Close
appWord.Application.Quit SaveChanges:=wdSaveChanges
```

```
Set docWord = Nothing
Set appWord = Nothing
'---------------------- suppression éventuelle du fichier mailing
strMessage = "Le fichier " & strNomFicMailing & vbCrLf & _
" contient les bons pour tous vos clients. " & vbCrLf & _
"Voulez-vous le conserver ?"
intchoix = ConFirmeOuiNon(strMessage)
If intchoix = vbNo Then
On Error Resume Next
Kill strNomFicFusion
Stage -VBA - les 28 & 29 Avril 1997 Page 6
Cours 08 - DIDIER Freddy
Kill strNomFicMailing
End If
EtiGauge.Caption = " Fin du traitement .... "
Exit_FusionWord97:
EtiGauge.Visible = False
CtlGauge.Visible = False
Exit Sub
Err_fusionword97:
AfficheMessage "Numéro erreur : " & Err & " description : " & Err.Description
Resume Exit_FusionWord97
End Sub
```

Explications sur le code

Nous allons commenter les instructions ligne à ligne.

Nous avons 4 variables de type chaîne strNomFicXxxxx qui contiendront respectivement le nom du fichier modèle, le nom du fichier de fusion, le nom du fichier mailing et enfin le nom de la base courante. Nous utilisons 2 variables entières pour mémoriser des choix.

Les lignes suivantes :

```
Dim docWord As Word.Document
Dim appWord As Word.Application
```

permettent de déclarer 2 variables objets, appWord pour lancer l'application Word, docWord pour utiliser un document dans Word.

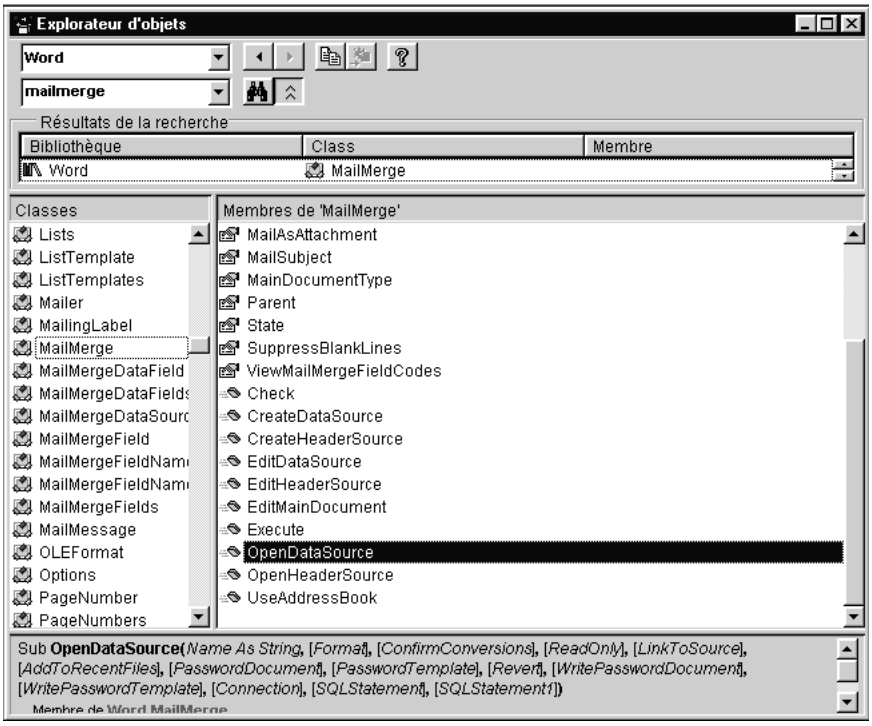
> *Si vous désirez que l'application serveur (ici Word) reste ouverte après l'exécution du sous-programme, vous devez déclarer Public les variables de type Object*

Par prudence, nous supprimons les anciens fichiers. S'ils n'existent pas, pour éviter un message d'erreur, nous utilisons le commande On Error Resume Next qui peut se traduire par En cas d'erreur, continuer à partir de l'instruction qui suit l'instruction ayant engendrée l'erreur.

> *Attention, il est important de rétablir une gestion normale des erreurs après l'utilisation de On Error Resume Next Nous effectuons une copie de notre fichier modèle (97BonAnniv-ori.doc). Nous assignons des valeurs à nos objets en utilisant l'instruction Set. L'application Word est rendue visible uniquement pour visualiser le traitement, le traitement peut se dérouler sans que Word ne soit visible. La fusion est réalisée grâce à la méthode MailMerge.*

> *Pour connaître les méthodes, les propriétés n'hésitez pas à recourir à l'explorateur d'objets. Dans un premier temps, la fusion ouvre la source des données en exécutant la procédure*

OpenDataSource. La fusion est dirigée dans un nouveau document, elle concerne tous les enregistrements de la source.



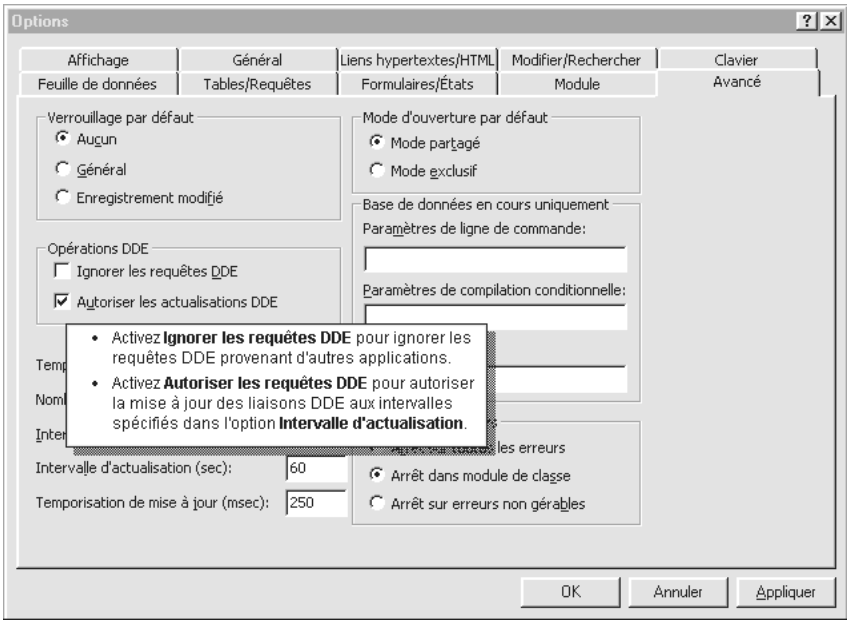
Le document actif de l'application Word est sauvegardé sous le nom BonAnniv.Doc.. Nous fermons l'objet docWord et lui assignons une nouvelle valeur (BonAnniv.Doc).

L'impression ne devant jamais être effectuée sans demander l'autorisation de l'utilisateur, nous lui posons la question. En cas de réponse positive, nous imprimons. Pour le stage, nous nous contenterons d'imprimer que la page courante . Puis le document est fermé,nous quittons l'application en sauvegardant d'office toutes les modifications. Il peut être intéressant de conserver le fichier Word créé. Donc nous posons la question à l'utilisateur.

Les transactions avec des objets OLE ne sont pas toujours commodes. Donc il faut être très patient et très prudent vis à vis de la gestion des erreurs.

La jauge fait partie des nouveaux contrôles appelés Active X. Pour insérer un contrôle de ce type, il faut sélectionner le bouton Autres contrôles dans la barre d'outils en mode création du formulaire.

Pour résoudre le problème du process Access ouvert par Word et non fermé, il faut cocher Ignorer les requêtes DDE dans l'onglet Avancé de la boîte de dialogue Options.



Attention, les modifications apportées dans cette boîte de dialogue resteront actives pour toutes les sessions ultérieures d'Access si elles ne sont pas rétablies. Voilà un problème qui peut nous attendre dans les salles de cours si des plaisantins modifient de manière inconsidérée les paramètres.

Le paramétrage peut être fait de manière logicielle. A titre d'exemple, regardez la procédure BasculIgnorerRequeteDDE() dans le module général MModifierOptions. Voilà, nous avons découvert le principe de l'automation.