BTS Informatique de gestion 1re année

Jean-Yves Février

Développement d'applications informatiques et génie logiciel

Cours 2 (ACCESS)

Directrice de publication : Valérie Brard-Trigo

Conseils généraux

Présentation du cours

Organisation du cours

Ce cours de programmation (Access) est constitué de 13 séquences et un devoir auto-corrigé. Je vous rappelle qu'il s'imbrique avec le cours d'analyse (Merise réf. 3932 TG 01).
Voici l'ordre dans lequel il faut exploiter tout cela :

- vous travaillez le cours d'analyse (Merise) jusqu'à la séquence 5 ;
- **tout en continuant l'analyse**, vous pouvez alors traiter le cours Access :
  - vous travaillez les séquences 1 à 13,
  - ensuite, vous faites le devoir auto-corrigé.

Attention, ne vous arrêtez pas à la séquence 5 du cours d'analyse. Il faut aller jusqu'à la séquence 5 pour avoir les prérequis nécessaires à l'apprentissage d'Access. Ensuite, vous devez travailler ces cours et le cours SQL (voir réf. 3999 TG 02) en parallèle.
Access est un outil très dense. En première année, nous étudions tout ce qui est utile aux deux options. En seconde année, nous l'approfondirons ensemble dans l'option développeur d'applications.

Choix d'Access et place du SGBDR

Introduction

Le cours que vous commencez vous permettra d'acquérir la pratique d'Access, le Système de Gestion de Base de Données Relationnel (SGBDR) semi-professionnel de Microsoft.
Par semi-professionnel, je n'entends pas semi-performant : c'est un produit très efficace s'il est utilisé à bon escient. Toute PME ou petite administration pourra utiliser des applications Access pour gérer ses données. En revanche, pas question pour EDF, une banque ou le Trésor Public de développer sous Access : quand on a des centaines de millions de données, il faut un outil professionnel d'une toute autre dimension : SQL Server, Oracle…
Pourquoi Access ? Nous avons hésité entre cet outil et d'autres du monde libre (MySQL, PosgreSQL). Finalement, nous avons retenu Access car c'est l'outil le plus répandu dans les entreprises où vous irez travailler (en action professionnelle, stage ou, ultérieurement, en vrai travail). Cela dit, l'essentiel de ce que nous allons étudier se transpose dans tout SGBDR.

Nous dirons donc que *nous allons apprendre un SGBDR par le biais de l'un de ses représentants, Access.*

Le SGBDR dans le référentiel

Comme je l'ai dit dans les conseils généraux de SQL, ce cours est indispensable pour le BTS Informatique de gestion car dans ce secteur d'activité, l'essentiel de votre travail consistera à manipuler des bases de données.
Dans le référentiel de la formation des deux options, l'étude d'Access se glisse un peu partout :

- explicitement dans le savoir S2 ALSI (Architecture logicielle des systèmes informatiques), précisément le savoir S25 du référentiel intitulé « Système de gestion de base de données relationnel » ;
- implicitement en S26 (architecture client-serveur) et S33 (Maquettage d'applications informatiques et programmation événementielle) ;
- également en S35 (Conception et développement d'applications à l'aide d'un langage de programmation procédural) ;
- pourquoi pas en S32 (Analyse et conception de systèmes logiciels) ? Oui, c'est le cours de Merise. Quel rapport avec Access ? Eh bien, la finalité d'une analyse Merise, c'est la création d'une application base de données !

Justifions tout cela : vous ne connaissez pas encore Access, mais je peux dès à présent vous dire que cet outil :

- est évidemment un SGBDR (d'où S25 et S26) ;
- possède des formulaires (feuilles de VB, formulaires de Delphi), d'où S33 ;
- possède le langage VB pour programmer votre application Access, d'où S35.

Ainsi, pour pouvoir exploiter pleinement Access, vous avez besoin de nombreuses connaissances cela en fait un outil d'informaticien d'une complexité sans commune mesure avec Word et Excel.
Nous n'allons pas étudier tout cela dans ce support : seul l'outil Access nous intéresse. Mais, comme je viens de le dire, pour pouvoir exploiter les formulaires, il faut connaître la programmation événementielle ; pour automatiser l'application, il faut savoir programmer…

Le SGBDR dans la formation

En formation initiale (en lycée), la première année s'étend sur 35 semaines, le volume horaire hebdomadaire de l'ALSI étant 2 heures de cours et 2 heures de travaux dirigés. Le SGBDR et SQL constituent la majeure partie de cet enseignement.
L'apprentissage d'Access est généralement réparti sur les deux années : le tronc commun en première année et le cours d'option en seconde année :

- en première année, on étudie tout… sauf la programmation avancée ;
- en seconde année (option *développeur d'applications*), on s'occupe de la programmation avancée.

Nous ferons la même chose dans ce cours.

Le SGBDR à l'examen

Access est un logiciel très intéressant pour mettre en œuvre les activités présentées dans l'épreuve de *pratique des techniques informatiques*, notamment en option développeur d'applications : les compétences C32, C33, C35 et C36 peuvent être bâties autour d'Access.
En option *administrateur de réseaux locaux d'entreprise*, les compétences C24 et C37 peuvent exploiter Access (quoique là, soyons honnêtes, installer Access pour la compétence *installer un SGBD* est un peu limité… sauf si l'on installe une architecture client/serveur par exemple).

Le SGBDR dans votre vie professionnelle d'informaticien

Dois-je justifier l'étude d'Access ? Ma foi, cet outil fait souvent partie des sujets de stage (option *développeur d'applications*) et sera certainement votre fond de commerce lors de votre arrivée dans la vie active.

Attention à ceux qui se sentent plus la fibre d'administrateur de réseaux locaux d'entreprise : vous avez l'impression qu'Access n'est utile que pour « ceux de l'autre option » ? Évidemment, seuls les développeurs d'applications présenteront Access à l'examen, mais ce sera sur les techniques de programmation avancée. Toute la partie que nous allons étudier cette année constitue un tronc commun que vous devez connaître : pour savoir installer et administrer une base de données, il faut savoir ce que c'est ! De plus, tout technicien supérieur en informatique de gestion doit pouvoir intervenir en maintenance ou en dépannage de premier niveau sur des domaines variés (configuration, installation, réseau, base de données) quelle que soit son option.

Présentation du support de cours

Principe

Ce cours a été conçu pour pallier au maximum les difficultés de l'apprentissage à distance : je l'ai rédigé de telle manière que sa lecture ressemble à un cours qu'un professeur étant en face de vous pourrait faire. Enfin, j'ai essayé autant que possible de faire cela.
Ainsi, à chaque difficulté potentielle ou interrogation que vous pourriez avoir, j'ai tenté de prendre les devants et de tout aplanir.
Mais j'insiste sur le point suivant : quelle que soit la qualité pédagogique de ce cours, il ne vous permettra pas d'assimiler Access par simple imprégnation visuelle. Vous devez fournir un travail d'apprentissage (le cours), de réflexion (toujours le cours) et d'entraînement (utilisation intensive d'Access).
Le début du cours comporte quelques exercices explicites, mais après, plus rien. Enfin… je n'ai pas pu les matérialiser comme dans le cours d'analyse ou de SQL, mais les exercices sont bels et bien présents : lorsque j'illustre une manipulation (et je n'arrête pas !), je vous dis systématiquement de la refaire vous-même et, si je ne le vous dit pas, c'est un oubli de ma part.
À la fin du cours, vous trouverez un sujet auto-corrigé d'application Access. La base corrigée est accessible à cette adresse :
http://www.campus-electronique.fr/bts-informatiquegestion puis en allant à la rubrique *téléchargement*. Jouez le jeu et ne vous précipitez pas sur cette dernière !

Organisation

Le fascicule de cours contient différentes choses :

- une ou plusieurs séquences par concept de base d'Access (tables, requêtes, formulaires, états, macros et modules) ;
- la séquence sur les modules est en fait une séquence sur la programmation sous Access : encore une fois, je vous donne des bases que j'espère solides, mais il n'est pas question de pousser les choses trop loin ;
- la dernière séquence est consacrée à l'ergonomie, c'est-à-dire à la façon de présenter correctement des applications. Ce n'est donc plus le côté technique qui est abordé, mais l'aspect visuel. C'est un thème qui m'est cher.

Notations

Pour vous aider à identifier les différents constituants du cours, j'ai utilisé les représentations suivantes :

- **tout ce qui est mis en couleur doit être appris par cœur.** Cela correspond aux définitions ou explications qu'il est absolument nécessaire de connaître pour s'en sortir en analyse. Quand je dis apprendre, ce n'est pas retenir pour l'heure qui suit afin d'épater les convives au prochain repas. Il s'agit d'une vraie leçon, dont vous devez vous souvenir tout au long de votre vie d'informaticien, ou, plus prosaïquement, au moins jusqu'à l'examen. Ces informations sont reprises à la fin de chaque séquence dans la fiche *synthèse* ;

Quelques conseils

Le seul conseil utile que je puisse vous donner est de garder à l'esprit la fable de La Fontaine *Le lièvre et la tortue* : il ne sert à rien de travailler ce cours comme un fou en juin ; travaillez-le plutôt dès maintenant quelques heures par semaine ré-gu-li-è-re-ment (j'aurais pu écrire **régulièrement** ou RÉGULIÈREMENT, mon but étant juste d'insister sur le mot).
La difficulté de l'enseignement à distance réside dans le fait que, par définition, vous êtes seul face au cours, personne n'étant là pour vous guider, insister sur l'essentiel ou établir la progression du cours.

Pour vous aider à suivre un rythme correct, disons que chaque séquence correspond à un travail d'environ une dizaine d'heures.

Attention à l'*environ* ! Vous avez sans doute le souvenir de vos études où, pour obtenir un même résultat, certains travaillaient toute la soirée et d'autres se contentaient d'être présents en cours. Il en est de même ici. Les 10 heures ne sont qu'un ordre de grandeur signifiant juste que 15 minutes, ce n'est pas assez, mais 30 heures, c'est trop.
Retenez qu'il vaut mieux passer 15 heures sur une séquence et la comprendre parfaitement, que faire exactement 600 minutes (10 heures) en passant à côté de l'essentiel.

Lorsque quelque chose ne vous semble pas clair, n'hésitez pas à creuser un peu, à étudier l'aide, à faire des manipulations par vous-même… il faut pratiquer !
La théorie que nous aborderons est essentielle car elle vous explique le sens des savoir-faire et vous aide à réagir aux problèmes.

Tout informaticien doit savoir faire. En fait, dans toute discipline, vous êtes recherché pour votre maîtrise des savoir-faire. Un comptable connaissant toute la théorie des optimisations fiscales mais incapable d'établir correctement une déclaration de revenus n'est pas un bon comptable. Si vous pouvez disserter une heure sur le protocole OSPF (administrateur réseau) ou sur le polymorphisme (développeur), mais que vous n'arrivez pas à configurer la connexion internet de l'entreprise, vous ne duperez personne.

Vous devez travailler chaque séquence la durée nécessaire à une réelle compréhension : quand vous comprenez le sens des manipulations présentées et que vous savez les refaire, c'est que la séquence est assimilée. J'insiste : vous devez tout refaire sur Access et faire des expérimentations.

Sommaire

Séquence 1

Prise en main d'Access

Dans cette séquence, nous étudions les différents concepts d'Access.

▶ *Capacités attendues*

- **Comprendre ce que vous pouvez faire avec Access**
- **Effectuer quelques paramétrages essentiels**
- **Savoir accéder à l'aide et aux fichiers base de données**
- **Utiliser une base Access avec une autre version d'Access**

▶ *Contenu*

1. Introduction

1A. Vocabulaire de base

1A1. Base de données

Une base de données (BDD) est un ensemble d'informations structurées se rapportant à un domaine précis (le système d'information vu dans le cours d'analyse).
Exemples :

- les informations permettant le suivi des stages en section de BTS ;
- les fiches d'état de stock dans un magasin.

Une base contenant toujours plusieurs données, il sera très mal vu d'omettre le pluriel au mot « donnée ».

1A2. Système de gestion de base de données relationnel (SGBDR)

Un Système de Gestion de Base de Données (SGBD) est un système informatique gérant l'ensemble des informations présentes dans une base de données.
Différentes façons de gérer les données ont vu le jour : les SGBD furent hiérarchiques puis réseaux ; aujourd'hui ils sont relationnels, demain ils seront objets (ou relationnels/objets). *Relationnel* fait référence aux relations mathématiques basées sur la théorie des ensembles cartésiens. On parle de SGBD Relationnels, soit SGBDR.

Il existe deux familles de SGBDR :

- les outils personnels convenant à la gestion des données de petites entreprises : Access, Paradox, MySQL… Leur intérêt ? Un rapport qualité/prix sans équivalent ;
- les systèmes professionnels, pouvant gérer un nombre considérable de données et d'utilisateurs. Citons Oracle, SQL Server, PosgreSQL… leur qualité ? La performance.

Ce support de cours concerne Access 2003. C'est un cours important car l'informatique de gestion exploite intensément les SGBDR ! L'obtention du BTS passe par la maîtrise d'un SGBDR, classiquement Access qui est très répandu.

1B. Étapes de développement d'une application

L'efficacité de l'application dépend fortement de votre connaissance de l'interconnexion des données, ce qui correspond au cours d'analyse (MCD). En pratique, un développement réussi (donc correct) ne peut se faire qu'en modélisant les données avant toute implantation sur ordinateur. En effet, un MCD inexistant ou faux impliquera une base de données inconsistante (c-à-d. erronée et inexploitable). Du MCD, vous déduisez le MLD qui décrira comment stocker les informations dans le SGBDR. La règle de développement suivante est donc impérative :

En d'autres terme s, vous devez être certain de la correction de vos MCD et MLD avant toute manipulation sur Access. Si, en cours de développement, vous êtes amené à corriger le MCD, l'ensemble du projet sera bien mal parti.
Pourquoi ce paragraphe ? Pour insister sur le fait que l'on ne doit surtout pas se lancer bille en tête dans la réalisation de l'application Access. Il faut préalablement conduire une analyse bien menée pour créer correctement les tables.

2. Les objets d'Access

La base de données contient les données de l'entreprise. Elle permet :

- de les stocker de façon rationnelle et efficace ;
- de les récupérer (exploiter) au mieux.

Access est un outil constitué des objets suivants :

- des **tables**, servant au stockage structuré des données ;
- des **formulaires**, chargés d'afficher les données et permettant également de les saisir ;
- des **requêtes** SQL (voir mon cours SQL !) permettant d'interroger la base, c'est-à-dire de récupérer les données ;
- des **pages**, qui permettent d'afficher et manipuler de l'information sur internet ; c'est l'équivalent du formulaire, mais orienté Web ;
- des **états**, dont l'objet est d'être imprimés. Ils fournissent les données de la base sous une forme lisible et permettent également (c'est le plus important) de produire les documents de l'entreprise à partir des données : les factures, les relances clients, les états de stock…
- le langage de programmation **VBA** (*Visual Basic for Applications*) qui permet de relier tous ces éléments pour en faire une application utilisable.

Nous allons étudier ces différents composants. Il ne s'agit pas d'être exhaustif, mais de débroussailler le terrain pour vous rendre autonome et capable d'aller plus loin.

3. Utilisation d'Access

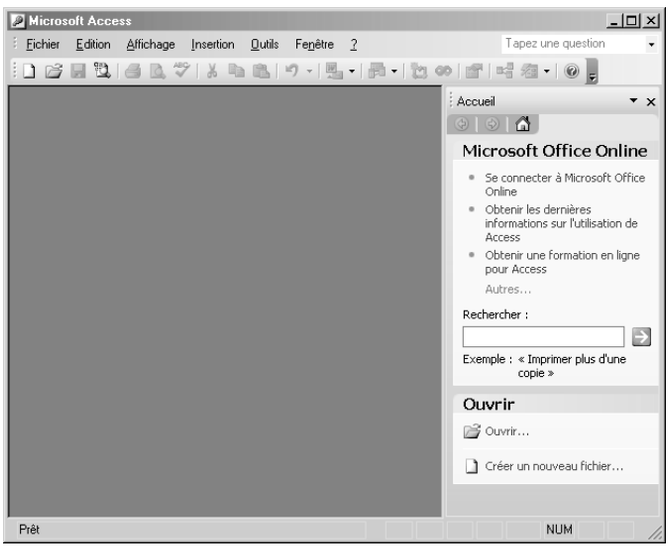
3A. Fichier

Concrètement, une base de données Access est un fichier. Cela signifie que la sauvegarde de la base revient à la sauvegarde d'un fichier.
L'extension du fichier base de données est *MDB* (pour *Microsoft Data Base*). En créant la base *Gestion*, vous générez un fichier nommé *gestion.mdb*.
La base de données est le réservoir d'informations de l'entreprise ; cette dernière y stocke ses clients, fournisseurs, commandes, factures… La base est donc un fichier crucial que l'on ne peut se permettre de perdre. La copie de sauvegarde de la base elle-même est une obligation professionnelle.

3B. Exécution d'Access

3B1. Paramétrage initial de l'interface

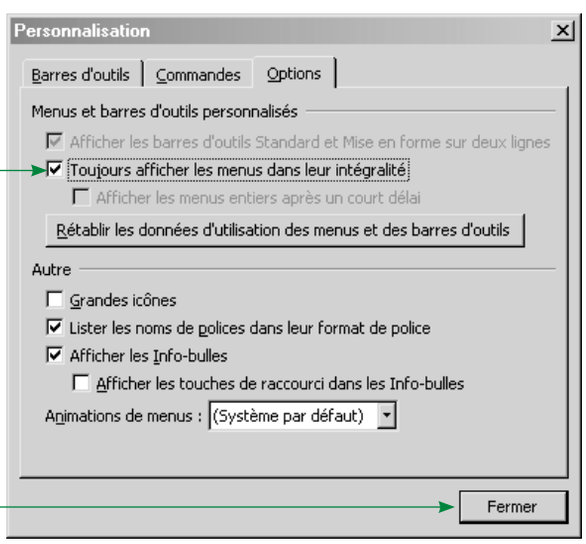
En lançant Access, vous obtenez sans doute la fenêtre suivante (si ce n'est pas le cas, c'est que votre paramétrage a déjà été modifié ; cela est sans importance).



Vous allez paramétrer les menus pour que toutes les commandes soient visibles. Pour cela, lancez la commande *Outils/Personnaliser…* ❶
Vous obtenez alors la boîte de dialogue suivante, à personnaliser comme moi.

L'essentiel, c'est que la deuxième case soit cochée.
Elle vous assure que les commandes ne joueront pas à cache-cache avec vous.
Si vous avez un problème pour activer la première case, cela peut être dû à un paramétrage contraire dans Word ou un autre outil du pack Office.

Quand vous avez fini, faites Fermer
Notez que cette boîte ne respecte pas l'ergonomie Windows car on devrait trouver des boutons OK *et* Annuler. *Ici,* OK *est remplacé par* Fermer et, pour annuler, il faut relancer la commande et restaurer les valeurs initiales à la main.

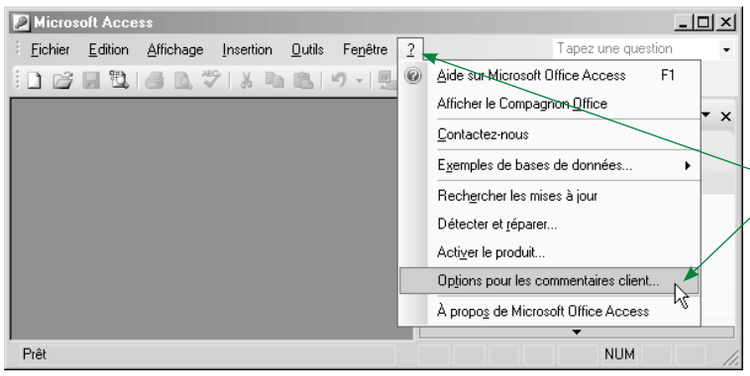


3B2. Paramétrage initial de l'aide

La version 2003 intègre une aide en ligne locale (sur votre PC) et une aide en ligne internet mise à jour continuellement (*Microsoft Office Online*). L'aide internet ne peut évidemment être utilisée que si vous disposez d'une connexion internet. De plus, je ne peux pas baser mon cours dessus puisse qu'elle évolue en permanence. Tous mes renvois vers l'aide dans le cours feront donc référence à l'aide locale, tout à fait complète d'ailleurs.

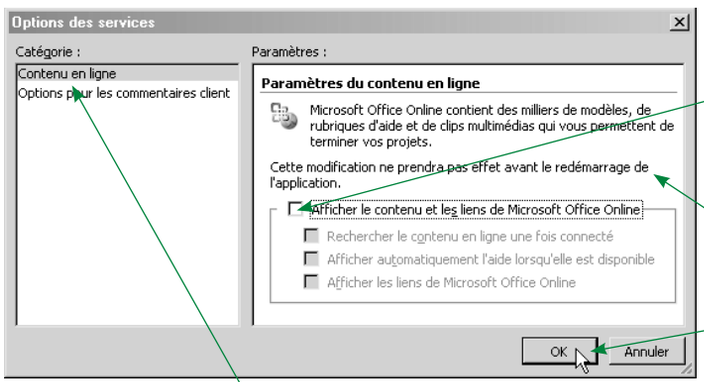
❶ *Je rappelle le sens de cette écriture conventionnelle : la commande* Outils/Personnaliser *signifie qu'il faut lancer le menu* Outils *(dans la barre de menus) puis lancer la commande* Personnaliser…

Pour éviter toute confusion, nous allons paramétrer votre version d'Access pour qu'elle désactiver l'aide internet. Voici la manipulation à réaliser :



Depuis la fenêtre Access, lancez la commande ?/Options pour les commentaires clients...

Vous obtenez alors cette boîte de dialogue :



Décochez alors la case Afficher le contenu et les liens de Microsoft Office Online si *elle était cochée.*

Notez qu'il faudra quitter puis relancer Access

Dois-je vous dire de cliquer sur OK *?*

Sélectionnez la catégorie Contenu en ligne.

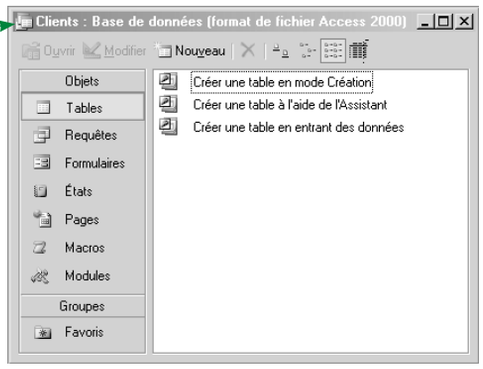
3C. Création d'une base et fenêtre Base de données d'Access

3C1. Création

J'ai choisi de créer une nouvelle base de données *Clients* à partir de l'interface Access. Le volet *Nouveau fichier* s'ouvre automatiquement. Je sélectionne alors *Base de données vide*. Access me demande immédiatement un dossier et un nom de fichier pour enregistrer la base. J'appelle ma base *Clients.mdb*.

J'obtiens la fenêtre suivante :

Notez cette information sur le format du fichier (voir paragraphe 3C3).



Ceci est la fenêtre de la base de données.
Tous ses composants (tables, requêtes…) sont accessibles (colonne de gauche, Objets*).*
Tout au long du développement de votre application, vous pouvez faire apparaître cette fenêtre en appuyant sur F11.

3C2. Le contenu de la fenêtre de la base de données

Vous remarquerez que la partie droite de la fenêtre contient des choses :

- « Créer une table… » pour les objets *Tables* ;
- « Créer un formulaire… » pour les objets *Formulaires*.

C'est étonnant, puisque ma base *Clients* vient d'être créée. Comme je n'ai encore rien mis dedans, je m'attendais naïvement à la trouver vide.
En fait, elle *est* vide. Nous voyons des raccourcis vers des assistants pour créer facilement de nouveaux objets. Pour voir (s'ils n'y sont pas) ou supprimer (s'ils y sont) ces assistants, il faut lancer la commande *Outils/Options…*❶, aller dans l'onglet *Affichage* puis cocher ou non la case *Raccourcis des nouveaux objets.* Je vous demande de **décocher** cette case.

3C3. Barre de titre de la fenêtre : format de la base

Le problème général de la compatibilité entre versions

La copie d'écran du paragraphe 3C1 attirait votre attention sur la barre de titre indiquant que ma base nouvellement créée était au format de fichier *Access 2000*. Quel est alors l'intérêt, je vous le demande, d'acheter la version 2003 d'Access si la base est au format 2000 ?

Des informations supplémentaires dans l'aide

L'aide en ligne (mode hors connexion) contient beaucoup d'informations sur les conversions et compatibilités entre les différentes versions Access (2.0, 95, 97, 2000, 2002 et 2003). Vous y apprendrez d'ailleurs qu'Access 2002 et 2003 utilisent le même format de fichier.
Pour accéder à tout cela, lancez l'aide❷ (*F1*) puis recherchez *convertir*. Le résultat intéressant parmi les vingt obtenus, c'est « À propos de la conversion d'un fichier Access ». Lisez-le !

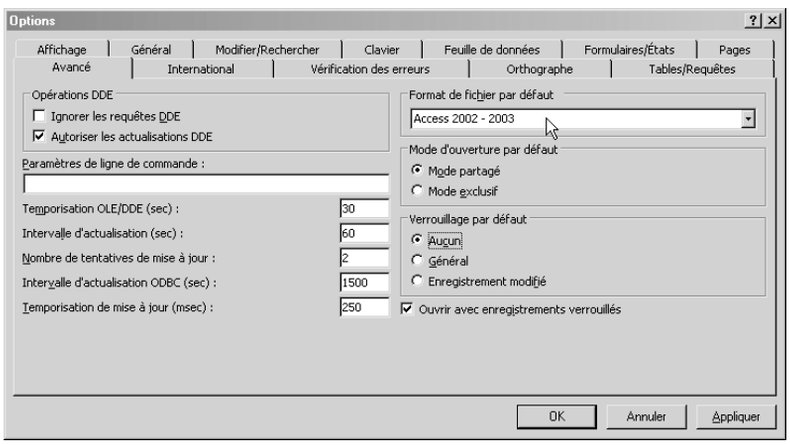
Pour conclure : quel format utiliser ?

Quitte à travailler sur Access 2003, nous utiliserons le format *Access 2002–2003*.

❶ *Pour accéder à la commande* Outils/Options*…, vous devez avoir une base de données ouverte dans Access.*

❷ *Je vous rappelle le paramétrage précédent pour n'exploiter que l'aide locale.*

Vous devez donc réaliser le paramétrage suivant. Il faut lancer la commande *Outils/Options…*❶. Or, *Options…* n'est actif que si un fichier Access est ouvert. Ainsi, alors même que le paramétrage ne concernera que les nouvelles bases et en aucun cas celle ouverte, il faut ouvrir une base quelconque pour accéder à la commande.
Bref. Une fois une base quelconque ouverte, lancez la commande *Outils/Options…* puis, dans la boîte de dialogue *Options*, sélectionnez l'onglet *Avancé* et choisissez *Access 2002-2003* dans la liste *Format de fichier par défaut* comme dans la copie d'écran ci-après :

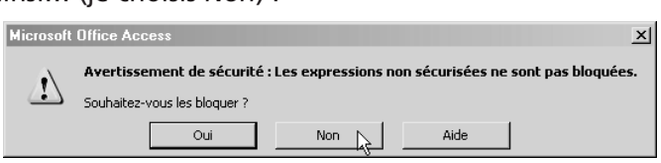


Ma base *Clients*, toujours à l'écran, reste comme convenu au format *Access 2000*.

3D. Ouverture d'une base existante et sécurité

Vous noterez qu'Access a un fonctionnement sensiblement différent des autres outils du *Pack Office* (Word, Excel et PowerPoint). Ces trois logiciels gèrent des objets (documents, classeurs ou présentations). Plusieurs peuvent être chargés en même temps. En revanche, Access ne peut traiter qu'une base à la fois. Si vous voulez ouvrir deux bases, vous aurez deux applications Access tournant simultanément❷. Cela est dû à la technicité des systèmes de gestion de base de données : ces outils sont, sans qu'il y paraisse, très pointus.
Lorsque j'ouvre une base existante (en double–cliquant sur son fichier d'extension *mdb* ou à partir de l'interface Access), j'obtiens des boîtes de dialogue bizarres.

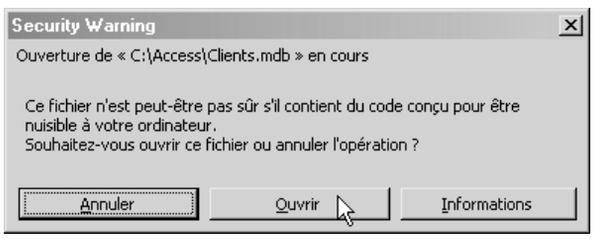
Cela commence ainsi… (je choisis *Non*) :



❶ *Petit rappel :* Outils/Options… *signifie menu* Outils *puis commande* Options…

❷ *En vrai, la programmation résout cela : si l'interface graphique d'Access ne permet de travailler que sur une base à la fois, vous pouvez utiliser du code VBA (liens ODBC par exemple) pour accéder aux données d'une autre base. Nous n'étudierons pas cela dans ce cours (c'est hors-programme). De plus, si vous avez besoin de réaliser ce genre de choses, c'est que votre développement commence à être dense… et qu'Access n'est plus forcément l'outil adapté.*

Cela continue comme cela (je choisis *Ouvrir*) :



Ces boîtes sont une nouveauté d'Access 2003. Elles concernent la sécurité de votre système qui peut être mise à mal par des applications nuisibles. La première boîte prend en compte le mode *sandbox*, la seconde concerne les virus macro.

Synthèse

Pour le moment, il y a assez peu de choses à retenir.

L'important est de bien distinguer le rôle du SGBDR Access vis-à-vis des autres produits du *Pack Office*.

De plus, il est crucial d'assimiler dès maintenant l'imbrication très forte de l'analyse et de la base de données : l'algorithmique est à la programmation ce que l'analyse est à la base de données. Je répète qu'il n'est concevable de passer à Access qu'une fois l'analyse conduite sérieusement.

C'est d'ailleurs un leitmotiv chez mes étudiants revenant de stage : « Oh, monsieur, vous aviez raison : j'ai commencé à faire la base immédiatement, mais j'étais ensuite incapable de réaliser les traitements demandés. J'ai dû recommencer en faisant l'analyse, ce qui m'a fort marri »❶.

Un dernier argument : avoir mené l'analyse correctement vous assure que vous avez bien en tête l'ensemble du système d'information que vous allez informatiser. Et cela, c'est déjà une base de départ solide.

Retenez enfin que quoi que vous fassiez (définition d'une table, d'un formulaire…), vous pouvez faire apparaître la fenêtre *Base de données* en appuyant sur *F11*. Cette fenêtre est indispensable puisqu'elle permet d'accéder à tous les constituants de l'application.

❶ *J'ai amélioré le style (mes étudiants ne parlent pas tous comme cela) mais l'idée est là.*

Séquence 2

Les tables 1/2 : origine et création

C'est dans les tables que les données d'une base sont stockées. C'est une séquence cruciale car la qualité de la définition des tables conditionne celle de l'application finale. Comme j'ai beaucoup de choses à vous dire (c'est simple mais c'est long), la présentation des tables se fera dans deux séquences :

- *dans cette séquence 2, nous verrons ce que sont les tables et comment les créer sous Access ;*
- *dans la séquence 3, nous détaillerons tout le paramétrage fin des champs constituant les tables.*

▶ *Capacités attendues*

- **Avoir parfaitement assimilé l'interaction entre l'analyse et la conception des tables la base de données**
- **Savoir créer des tables de façon rigoureuse (c'est-à-dire en exploitant correctement les propriétés des champs)**

▶ *Contenu*

1. Introduction

Nous allons voir comment définir des tables puis comment les remplir.

La base de données que vous allez mettre en place sera composée de tables. Une table, au sens informatique, est un ensemble d'informations se rapportant à un sujet particulier. Les données d'une table sont présentées sous la forme d'un tableau comportant des lignes (appelées *enregistrements*) et des colonnes (appelées *champs*). Les tables (leurs champs) sont définies dans le MLD.
Ainsi, je vous rappelle l'obligation de faire un MCD puis un MLD avant de commencer le moindre développement sur Access. Ce n'est pas pour vous brimer, mais parce qu'il est tout simplement impossible, pour vous comme pour moi, d'arriver à faire de tête la définition des tables pour un cas un tant soit peu intéressant. Bien entendu, si vos tables sont mal définies, vous ne représentez pas correctement la réalité. Votre application Access ne pourra donc pas réaliser ses objectifs.

Nous avons vu dès le cours SQL que le monde de l'analyse possède un vocabulaire différent de celui des bases de données. Le tableau suivant rappelle les différents termes :

Vocabulaire MCD	Vocabulaire MLD et SGBD	Vocabulaire SQL
identifiant	**clé primaire**	champ
propriété	**champ**	
occurrence	**enregistrement**	enregistrement
entité ou association	**table**	table

Sans refaire de démonstration comme dans le cours d'analyse, je vous rappelle juste que la différence de nom n'est pas aussi arbitraire et lourde qu'il y paraît : les concepts sont voisins mais pas tout à fait identiques puisque chaque univers (l'analyse et la base de données) apporte sa structuration.

2. Création des tables

Il y a trois façons de créer une table. Il est possible :

- d'utiliser un atelier de génie logiciel (AGL) qui, partant du MCD, va générer le MLD puis les tables dans Access. Vous n'avez alors rien à faire sous Access mais il n'y a pas de miracle : ce que vous ne ferez pas sous Access, vous devrez le faire avec l'AGL. Nous verrons cela dans la séquence 5 ;
- d'exécuter des requêtes SQL du langage de définition des données (voir le support SQL, séquence 8) ;
- de définir les tables à la main depuis la fenêtre *Base de données*. Il s'agit alors simplement de donner la liste des champs avec leur type et de définir une clé primaire. C'est ce que nous allons voir maintenant.

Créer les tables ne revêt aucune difficulté. Il suffit de le faire. C'est long, pas très marrant, mais facile et indispensable. C'est en effet une étape cruciale où les malfaçons se payent très cher. En effet, si vous définissez mal un champ, vous devrez revoir tous les formulaires et le code où il apparaît pour les corriger. Plus vous en êtes aux fondations de votre application, plus les erreurs sont coûteuses à corriger. D'où l'importance d'un MCD et d'un MLD corrects.

2A. Introduction

C'est la façon la plus naturelle pour un projet de taille normale où vous ne disposez de toute façon pas d'un AGL.
Nous allons créer la base de données *Bibliothèque* contenant deux tables définies ainsi dans le MLD :

Livre (NumLivre, Titre, NbrPages, DateAchat, PrixAchat, NumAuteur#)

Auteur (NumAuteur, NomA, PrénomA, DateNaissanceA)

Légende : clé primaire, clé étrangère#

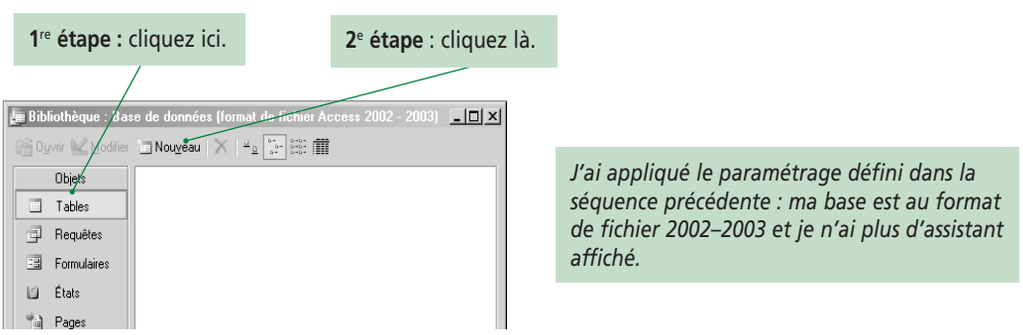
Pouvons-nous nous lancer bille en tête ? Non, bien sûr, car si nous avons le nom des différents champs, nous n'avons pas leur type. Il faudrait en théorie se référer au dictionnaire des données pour les avoir. J'ai bien dit *en théorie* car ici les noms des champs indiquent explicitement leur type.

2B. On y va !

Pour créer la table, c'est simple ! Dans un premier temps, créez une base de données *Bibliothèque*❶. Ensuite, faites les deux étapes suivantes :

1. dans *Objets*, sélectionnez (cliquez sur) *Tables;*

2. dans la barre d'outils, cliquez *Nouveau*.



J'ai appliqué le paramétrage défini dans la séquence précédente : ma base est au format de fichier 2002–2003 et je n'ai plus d'assistant affiché.

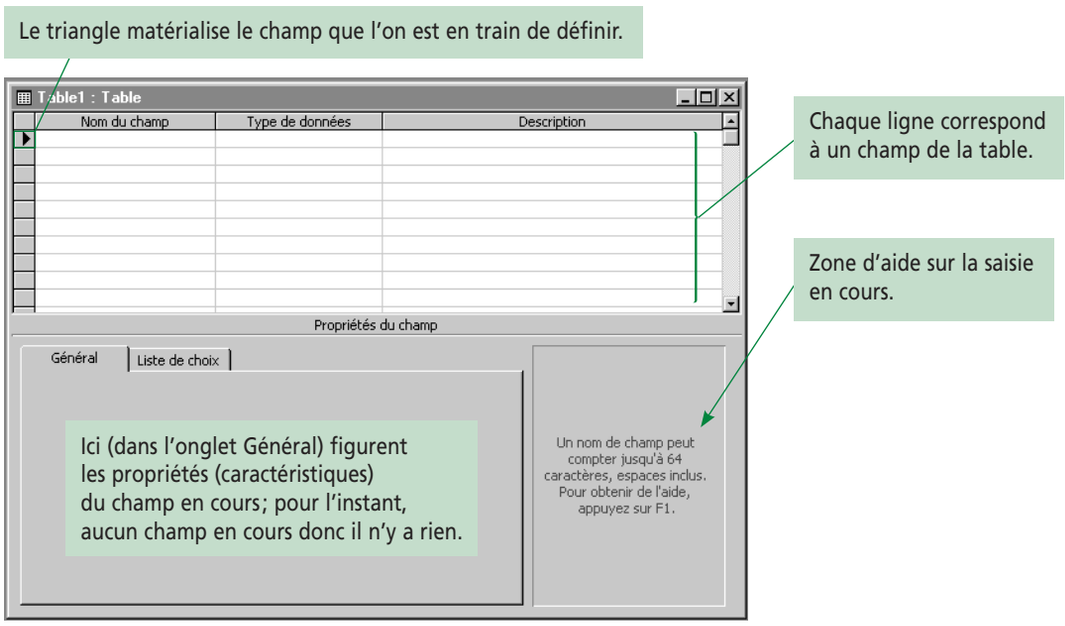
Avant de continuer, observez la copie d'écran ci-dessus et comparez-la avec la fenêtre base de données que vous avez sur votre écran.
Notez que vous avez accès à trois boutons :

- *Nouveau* crée une table. C'est cette commande que vous devez utiliser ;
- *Modifier* modifie la structure d'une table : vous pouvez ajouter, modifier ou supprimer des champs. Comme aucune table n'est sélectionnée (et pour cause, la base n'en contient pas), cette commande est inactive ;
- *Ouvrir* permet de visualiser, ajouter ou modifier les données d'une table (ouverture de la table en mode *Feuille de données*). Comme aucune table n'est sélectionnée, cette commande est inactive.

Une fois *Nouveau* sélectionné, vous obtenez un menu proposant cinq choix possibles. Cliquez sur les différentes propositions dans la partie droite. La partie gauche vous donne alors un (bref) descriptif.

❶ *Vous avez appris à faire cela dans le paragraphe 3C de la séquence 1. Je ne peux que vous conseiller de créer un dossier (du genre* Z:\BTSIG 1re année\Alsi\Access*) pour stocker vos différentes bases de données.*

Choisissez *Mode Création* et validez. Vous obtenez alors la fenêtre suivante :



Vous remarquerez que nous allons rentrer les différents champs dans un tableau❶ (un par ligne). Chaque champ sera caractérisé par trois informations :

- son **nom** (obligatoire). Il peut être composé de caractères quelconques, dont des espaces. Cela dit, utiliser des espaces rendra l'écriture des requêtes et des formulaires plus complexe. Je vous recommande de n'écrire le nom du champ que sous la forme d'un mot et de mettre par exemple *DateLecture* au lieu de ~~*Date de lecture*~~;
- son **type** (obligatoire), à savoir le type des données qu'il contiendra. La saisie est aisée puisqu'une liste déroulante vous propose les différents types possibles;
- une **description** (facultative). C'est une phrase en français explicitant si besoin est la signification du champ.

2C. Les types des champs

Remplissez le tableau pour avoir la même chose que moi. Lors de la définition des différents types de champs, observez bien le contenu de l'onglet *Général* en bas à gauche dans la fenêtre; il indique les différentes propriétés du champ courant. N'hésitez pas à utiliser *F1* quand la zone d'aide vous le propose pour lire la description des différents types et propriétés.

(il est clair que cette table permet de stocker des informations sur des livres.)

❶ *Access se définit par Access (de même qu'un compilateur VB peut être écrit en VB). Ainsi, les descriptions des tables que vous créez sont stockées dans des tables* système. *Pour le voir, menu* Outils/Options…, *onglet* Affichage, *rubrique* Afficher, *cochez* Objets système. *Décochez cette case après avoir vu les tables.*

Vous remarquerez que les types ne sont pas définis très finement : on ne distingue par exemple pas les entiers des réels… pour le moment ! Cela se fera dans l'onglet *Général*. Pourquoi *NumLivre* est-il en *NuméroAuto* et *NumAuteur* en *Numérique* ? Nous allons l'expliquer dans quelques lignes.
Voici un résumé des différents types de données. Pour plus d'explications, cherchez dans l'aide en ligne !

Type de données	À quoi correspond le type	Remarque ou exemples de valeur
Texte	Caractères alphanumériques (tous les caractères du clavier). C'est le type par défaut proposé par Access. Il permet de stocker 255 caractères au maximum	"Bonjour !" "Alice au pays des merveilles"
Mémo	Quelques phrases ou paragraphes (65 536 caractères maximum)	Pour stocker du texte, utilisez *Texte*. Si la longueur dépasse 255 caractères, utilisez *Mémo*.
Numérique	Valeur numérique (entière ou réelle)	12 4,56
Date/Heure	Date et heure (ou que date, ou que heure)	07/12/2008 12:30:00 dimanche 7 décembre 2008 le choix d'affichage (date et/ou heure, format) sera défini dans un second temps.
Monétaire	Valeur monétaire (numérique) (avec le symbole € si les paramètres régionaux de Windows sont corrects)	12,19 €
NuméroAuto	Valeur numérique unique servant d'identifiant. *(Voir ci-dessous plus d'explications.)*	C'est soit une valeur aléatoire tirée au sort par Access, soit une valeur qui s'incrémente de un en un comme un compteur. Access vous assure que vous n'aurez jamais deux fois la même valeur. Pratique pour définir simplement un identifiant !
Oui/Non	Valeur booléenne : soit *oui* soit *non*	Pour stocker des informations logiques (dont la valeur est *oui* ou *non*). Par exemple *marié*, *décédé*…
Liaison OLE	Objets OLE, graphiques et autres données binaires	Bof, bof. Nous ne jouerons pas avec cela.
Lien hypertexte	Texte considéré comme un lien hypertexte : il sera souligné d'une couleur particulière et si vous cliquez dessus, vous accéderez à la ressource correspondante	La valeur est du texte. Mais choisir ce type de données, c'est dire à Access de considérer la valeur comme étant un lien. Si la valeur est un lien valide, tout se passera bien. Sinon, vous aurez une erreur en cliquant dessus.
Assistant liste de choix	La valeur à saisir sera choisie dans une liste de valeurs issues d'une autre table ou rentrées en dur. *(Voir ci-dessous plus d'explications.)*	

Le type Liste modifiable *est un peu particulier puisqu'il définit l'origine des données plutôt que leur type. Nous l'étudierons dans la séquence sur les contrôles.*

Nous pouvons maintenant dire pourquoi *NumLivre* est en *NuméroAuto* et *NumAuteur* en *Numérique*. Tous nos identifiants artificiels seront des numéros automatiques. Ces numéros sont des valeurs entières. Or, dans *Livre*, *NumAuteur* est une clé étrangère.

Ses valeurs viennent donc d'un champ *NumAuteur* clé primaire dans une autre table (la table *Auteur* indiquée dans le MLD paragraphe 2A). La clé primaire *NumAuteur* sera donc *NuméroAuto*, tandis que la clé étrangère *NumAuteur* sera numérique.

Concernant les types de données, j'aimerais faire une remarque importante : comment stocker un code postal ou un numéro de téléphone ? Ces informations sont des chiffres. Il semble donc logique de les stocker en valeurs numériques. Cependant, vous savez d'après votre cours d'architecture matérielle que les valeurs numériques prennent plus de place en mémoire que le texte. Or, quel est le seul intérêt des valeurs numériques ? Les calculs que l'on peut faire avec ! Et alors, je vous le demande solennellement : quels calculs allez-vous faire avec les codes postaux ou les numéros de téléphone ? Additionner des codes postaux, calculer la moyenne de numéros de téléphone… cela n'a pas de sens ! Stocker ces données comme des chiffres n'apporte donc rien.

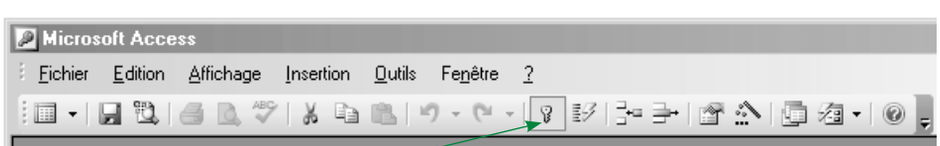
En revanche, les stocker comme du texte est intéressant car cela apporte :

- gain de place ;
- facilité de contrôle de la saisie (le code postal comporte cinq caractères (des chiffres), le numéro de téléphone en contient dix) ;
- facilité de manipulation : pour avoir tous les clients habitant le Nord (département 59), on demandera les clients dont le code postal commence par les deux caractères *59*. Vous savez que les langages de programmation possèdent beaucoup de primitives permettant la manipulation des chaînes.

En conclusion, vous emploierez au maximum le type de champ *texte*, même si les données ne contiennent que des chiffres. En pratique, vous n'utiliserez les types numériques que lorsque vous aurez réellement besoin de les manipuler en tant que chiffres, donc lorsque vous aurez des calculs à faire.
Enfin, toutes les données contenant des montants monétaires (prix, taux de TVA…) seront définies en type monétaire. Ce type est avant tout un type numérique donc il permet les calculs. Son intérêt ? Il possède deux décimales et utilise le symbole monétaire défini dans les paramètres régionaux de Windows. Notez que le type *monétaire* est simplement un type réel à deux décimales avec un format d'affichage particulier. Vous pourriez donc le définir vous-même avec les propriétés que nous étudierons dans la séquence suivante. Si ce type existe en standard, c'est pour vous éviter d'avoir à le créer sachant qu'il est très fréquent en informatique de gestion.

2D. Spécifier les clés primaires (et étrangères ?)

Rien ne nous dit que *NumLivre* est clé primaire. Le fait d'utiliser le type *NuméroAuto* n'est pas suffisant. En effet, ce type est un type comme un autre pour Access. Comment allons-nous donc faire pour spécifier la clé primaire ?
En fait, c'est très simple : vous vous placez sur la ligne du champ que vous voulez définir en clé primaire puis clic droit et vous choisissez *clé primaire* (ou vous cliquez sur la clé dans la barre d'outils *Création de table* qui doit être visible).



Bouton *clé primaire* dans la barre d'outils *Création de table*.

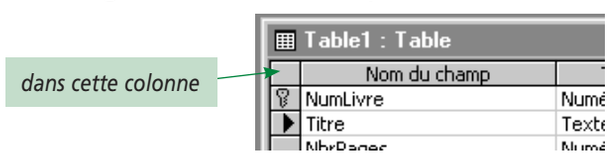
Définissez *NumLivre* en clé primaire. Une petite clé apparaît alors à gauche du champ pour matérialiser sa nouvelle condition :

Si vous refaites cette manipulation en restant sur *NumLivre*, la clé primaire disparaît. C'est donc une bascule qui place ou enlève la clé primaire.

Si vous voulez définir un autre champ en clé primaire, pas de problème, refaites la manipulation sur un autre champ et la clé primaire sera automatiquement supprimée de *NumLivre* pour être basculée sur le nouveau champ.

Vous noterez que dans l'onglet *Général*, la propriété *Indexée* de *NumLivre* change : elle sera *Oui – Sans doublons* si le champ est clé primaire et *non* sinon. Utilisez l'aide (placez-vous dans la zone de saisie de *Indexé* et faites *F1*) pour des détails.

Comment faire pour définir une clé primaire constituée de plusieurs champs ? Par exemple, pour une table issue d'une association du MCD : vendre (NumProduit#, NumFournisseur#). C'est simple, il faut sélectionner les champs en groupe.
Pour cela, placez votre curseur souris dans la colonne contenant le triangle et la clé (juste à gauche des champs), sur la ligne de l'un des champs concernés.



Le curseur souris devient alors une flèche (➔). Cliquez : la ligne entière est sélectionnée. Placez votre curseur sur une autre ligne, toujours dans la même colonne. Faites *Ctrl+Clic*❶. Les deux lignes seront sélectionnées en même temps. C'est exactement le principe de la sélection multiple de Windows.

Une fois que toutes les lignes contenant les champs devant faire partie de votre clé primaire sont sélectionnées, vous pouvez cliquer droit pour sélectionner *Clé primaire* (ou clic sur le bouton dans la barre d'outils).

Par exemple, j'ai sélectionné les trois champs *Titre*, *DateAchat* et *NumAuteur* et je les ai définis comme constituant ma clé primaire (cela n'a pas de sens mais la question n'est pas là). On voit plus ou moins les petits symboles clé primaire :

Vous remarquez que NumLivre *n'est plus clé primaire.*

❶ *Maintenez la touche* Contrôle *appuyée et cliquez.*

Faites la manipulation puis revenez dans la configuration standard (*NumLivre* est la seule clé primaire).
Que nous reste-t-il à faire ? Indiquer que *NumAuteur* est une clé étrangère. Bon, c'est vite fait : les clés étrangères ne s'indiquent pas ici, mais avec les relations (voir la séquence 4).

Avant de définir plus finement les champs, nous allons voir comment rentrer les données. Fermez la table. Access vous demande si vous souhaitez l'enregistrer. Dites *Oui* et appelez-la *Livre*.

Remarque
Si vous enregistrez une table sans clé primaire, Access vous propose d'en ajouter une. Si vous acceptez, Access crée un champ *N°* de type *NuméroAuto*.

2E. Faites une table vous-même !

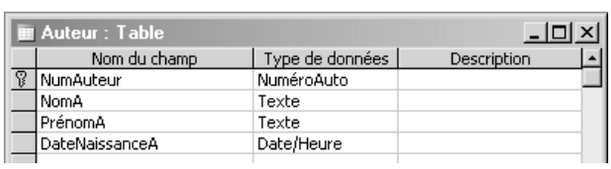
Exercice

En appliquant ce que nous venons de voir, créez la table *Auteur* dont je vous rappelle le MLD :

Auteur (NumAuteur, NomA, PrénomA, DateNaissanceA)

Utilisez les bons types de données (vous aurez besoin de *Date/Heure*, *Texte* et *NuméroAuto*).

Voilà une copie écran de la solution (sans surprise) :



Synthèse

Que retenir de cette séquence ? Et bien, d'assez nombreuses choses !

La table est constituée de champs qu'il faut décrire très précisément par :

- leur type, à savoir le type des données contenues dans le champ ;
- la clé primaire.

Il ne faut pas oublier que si la clé primaire est de type *NuméroAuto*, les clés étrangères associées doivent être *Numérique/entier long*. D'une façon générale, clés primaire et étrangère seront de même type : deux nombres ou deux chaînes par exemple.

Toutes les manipulations vues doivent être assimilées.

Séquence 3

Les tables 2/2 : propriétés des champs

C'est dans les tables que les données d'une base sont stockées. C'est une séquence cruciale car la qualité de la définition des tables conditionne celle de l'application finale. Comme j'ai beaucoup de choses à vous dire (c'est simple mais c'est long), la présentation des tables se fait dans deux séquences :

- *dans la séquence 2, nous avons vu ce qu'étaient les tables et comment les créer sous Access ;*
- *dans cette séquence, nous allons détailler tout le paramétrage fin des champs constituant les tables.*

▶ *Capacités attendues*

- **Vous devez savoir créer un champ de façon rigoureuse (c'est-à-dire en exploitant correctement les propriétés des champs)**

▶ *Contenu*

1. Définition fine des champs

1A. Intérêt des propriétés des champs

Il s'agit de travailler sur l'onglet *Général* qui apparaît en bas à gauche dans la fenêtre de création de la table. Cet onglet permet de définir finement les propriétés (caractéristiques) d'un champ en fonction de son type. Nous avons vu ci-dessus que :

- les champs pouvaient avoir une valeur par défaut. Par exemple, le prix d'achat des livres était par défaut de 0,00 €. Quel est l'intérêt d'une valeur par défaut ? Si vous saisissez habituellement les livres le jour de leur achat, vous mettrez la date du jour en valeur par défaut pour *DateAchat*. Vous n'aurez donc pas à la saisir. Le jour où vous entrez les livres le lendemain de l'achat, vous saisirez la date de la veille à la place de la valeur par défaut. L'intérêt de cette dernière est de limiter le travail de saisie. On gagne du temps et on réduit le risque d'erreur.
  Cela dit, un livre ne coûte jamais 0,00 €. Une telle valeur par défaut n'est donc pas judicieuse puisqu'il faudra toujours la modifier. En fait, elle a été mise automatiquement par Access. Il faudra la supprimer ;
- les champs pouvaient ne pas avoir de valeur (la date de naissance de Maupassant n'était pas renseignée). Il est parfois utile d'avoir la possibilité de ne pas donner de valeur à un champ, par exemple quand :
  – on n'a pas l'information (un client n'a pas voulu donner son numéro de téléphone),
  – l'information n'existe pas (un auteur vivant n'a pas de date de décès définie).

Pour autant, certains champs doivent impérativement avoir une valeur ; par exemple, une facture dont l'identité du payeur n'est pas indiquée… c'est grave !
Vous l'avez deviné, c'est au travers des propriétés de chaque champ que l'on précisera une éventuelle valeur par défaut, l'obligation ou non de fournir une valeur…

En fait, les propriétés fines des champs vont permettre :

- d'affiner leurs caractéristiques (un code postal doit tenir sur cinq caractères, la longueur par défaut de 255 caractères n'est donc pas adaptée) ;
- d'augmenter la lisibilité des informations affichées (propriétés *légende*, *format*) ;
- d'aider à la saisie des informations, ce qui limite le risque d'erreurs et permet d'aller plus vite (*masque de saisie*, *valeur par défaut*, *valide si…*).

Bref, ces propriétés ont avant tout pour objet d'améliorer l'ergonomie de l'application. Il est un peu fastidieux de les définir mais il ne faut surtout pas sauter cette étape : la qualité de la base en dépend. Nous allons énumérer les différentes propriétés.

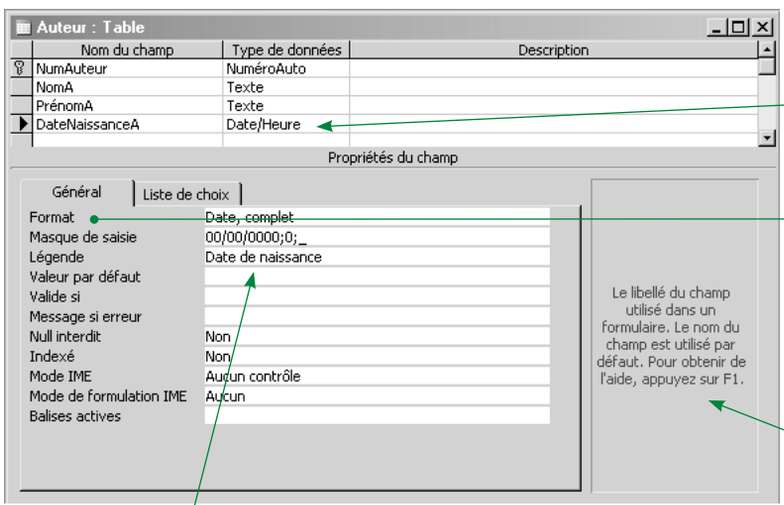
1B. Présentation des propriétés

Chaque champ possède différentes propriétés dépendant de son type (sa longueur pour du texte, son nombre de décimales pour une valeur numérique…). Lorsque vous créez un champ, Access donne des valeurs par défaut à ses propriétés ; ces valeurs ne seront pas toujours adaptées à la sémantique du champ.
Par exemple, un champ *texte* sera par défaut de longueur 50❶. S'il représente un nom de famille, on pourrait se limiter à 20 caractères (économie) ; en revanche, si le champ contient un libellé ou une description, on peut trouver les 50 caractères alloués un peu justes.

❶ *Paramétrage modifiable par la commande* Outils/Options…, onglet Tables/Requêtes, cadre *Taille du champ par défaut.*

Il est très important de définir avec précision les propriétés des champs d'une table, cela vous permettra de gagner un temps précieux par la suite. En effet, la génération automatique de formulaires s'appuie sur la définition complète des champs ; si tout n'est pas renseigné ou si les propriétés ne sont pas correctes, il faudra modifier manuellement les champs des formulaires.

Pour définir une propriété de champ dans une table, suivez les étapes ci-dessous. (Je reprends la table *Auteur* pour illustrer mon propos.)



1re étape : le champ à définir doit être le champ courant. Ici, je vais définir *DateNaissanceA*.

2e étape : vous pouvez alors définir les différentes propriétés du champ. Elles dépendent de son type.

Cette zone explicite la propriété courante (ici, *Légende*). **On vous rappelle que l'aide est là. Profitez-en !**

On ne voit pas le curseur, mais je suis en train de saisir la propriété *Légende*.

J'insiste : les propriétés dépendent du type du champ. Certaines, comme *Légende*, sont communes à tous les types, d'autres sont propres à un seul. Par exemple, le nombre de décimales n'a de sens que pour le type numérique réel et pas pour le type texte.

Vous trouverez à la page suivante, les propriétés des types principaux (dans l'ordre, *Texte*, *Date/Heure*, *Numérique*, *Monétaire* et *NuméroAuto*). Il ne s'agit pas de les apprendre par cœur, mais de savoir les utiliser correctement lorsque l'on en a besoin. Ne tenez pas compte des valeurs attribuées à certaines propriétés, ce sont les valeurs par défaut d'Access.

La propriété **Format** *du type* Monétaire *propose notamment la valeur* Euro *(pour utiliser l'unité* euro*) ou* Monétaire *pour utiliser l'unité monétaire mentionnée dans les paramètres régionaux de Windows (logiquement l'euro également pour nous).*

Nous allons étudier les différentes propriétés. Pour une explication plus détaillée, cliquez dans leur zone de saisie et faites *F1*. Je suis le premier à me servir de l'aide lorsque je veux définir correctement les champs.

2. Étude des différentes propriétés techniques

Je n'étudierai pas les propriétés dans l'ordre où elles sont proposées dans l'interface. J'ai préféré les regrouper par thème. Cela ne change évidemment rien !
Les propriétés techniques❶ sont avant tout destinées à Access. Elles permettent le paramétrage et la manipulation correcte des données.

2A. *Propriété Taille du champ*

Comme toute variable en programmation, un champ est d'un type précis qu'il faut indiquer lors de la conception de la table. En programmation, nous définissons complètement le type dès la déclaration. En effet, on peut indiquer la longueur de la chaîne, le type d'entier (*byte*, *octet*, *integer*…) ou de réel.
Sous Access, il faut dans un premier temps définir le type général (numérique, texte…) et, dans un second temps, préciser le « sous-type » avec la propriété *Taille du champ*.

❶ *Ce n'est pas un terme technique. Comme il y a beaucoup de propriétés, je les ai classifiées selon leur objet pour des raisons pédagogiques.*

Vous remarquerez les valeurs par défaut :

- un champ numérique sera *Entier long*. Vous disposez d'autres sous-types entiers, mais aussi réels ;
- un champ texte fera 50 caractères (le maximum étant 255).

Rappel : une clé étrangère liée à une clé primaire définie en *NuméroAuto* doit être *Entier long*.

Cette propriété est importante pour les champs *Texte* et *Numérique*. En limitant la taille des champs, vous garantissez que les données saisies ne dépassent pas une certaine longueur. Inversement, préciser la longueur du champ évite de perdre de la place (utiliser 50 caractères pour stocker un nom est trop généreux). Les tailles sont les suivantes :

- pour les champs *Texte*, la taille proposée par défaut est de 50 et peut être comprise entre 1 et 255 ;
- pour les champs *Numérique*, la taille détermine la fourchette des valeurs autorisées. Le tableau ci-dessous présente ces différents intervalles ainsi que la place consommée pour stocker la valeur. C'est une recopie de l'aide (sauf que l'aide donne les noms de type en anglais ; j'ai repris les noms français que vous trouverez dans la liste déroulante de la propriété *Taille du champ*) :

Type	Intervalle de valeurs	Décimales	Taille de stockage en octets
Octet	0 à 255	0	1
Entier	-32 768 à 32 767	0	2
Entier long (par défaut)	-2 147 483 648 à 2 147 483 647	0	4
Décimal	de -10^{38} à 10^{38} (en gros)	28	12 (douze !)
Réel simple	$-3{,}4\times10^{38}$ à $3{,}4\times10^{38}$	7	4
Réel double	$-1{,}797\times10^{308}$ à $1{,}797\times10^{308}$	15	8

Vous noterez la palette de choix : on peut aller jusqu'à 10^{308} soit *1* suivi de 308 zéros[1] avec le réel double. Si l'on veut de la précision, on a le type *décimal* avec 28 décimales. Notez que plus on a de chiffres, plus cela prend de place en mémoire : là où vous stockez un décimal, on peut mettre douze octets.
Il est conseillé d'utiliser la plus petite définition de taille possible afin d'optimiser les temps d'accès :

- pour un numéro de rue, une taille *entier* suffit (certes, les valeurs négatives ne servent pas, mais enfin… le type de données « entier positif » de 0 à 65 535 n'existe pas sous Access) ;
- pour un âge, *octet* est idéal. Laisser *Entier long* (la valeur par défaut) consommerait 4 fois plus de mémoire pour rien.

Ne tombez pas dans l'excès contraire en allouant des tailles trop parcimonieuses :

- un numéro de rue défini comme *octet* posera un jour ou l'autre des problèmes ;
- un nom codé sur dix caractères n'est pas une bonne idée.

Il faut bien comprendre que la définition de la taille (et du type) doit être valable pour tous les enregistrements : si une table contient 100 000 enregistrements, que pour 99 999 de ceux-ci la valeur du champ *X* est codée sur 2 caractères et que pour l'enregistrement restant elle l'est sur 10, il faudra définir *X* sur 10 caractères (ou, de façon plus efficace,

[1] *De mémoire, je crois que le nombre d'atomes dans l'univers est de l'ordre de 10^{30}. Or, un atome, c'est petit, et l'univers, c'est grand.*

revoir le codage de ce champ). La définition sera donc faite en fonction de l'occurrence supposée la plus grande.

2B. Propriété Null interdit

La valeur *Null* est une valeur prédéfinie (constante) qui signifie que le champ est vide (ne contient rien). Ainsi, si vous créez un enregistrement, les champs qui ne seront pas remplis auront la valeur *Null*. Vous pouvez ensuite tester sous SQL avec l'opérateur *is* si un champ contient ou non une valeur (voir le paragraphe 6B de la séquence 4 du cours SQL).

Pour qu'un champ puisse contenir la valeur *Null*, vous devez affecter la valeur *non* à la propriété *Null interdit*. Je vous conseille de prendre cela en compte dans la règle de validité (paragraphe 3E) et d'autoriser soit la valeur nulle, soit une valeur valide. Vous l'écrirez donc sous la forme « Est Null ou … » (dans *Est Null* vous retrouvez la forme francisée du *is Null* de SQL). Ce n'est qu'un conseil car Access accepte que vous n'incluiez pas *Null* dans la condition. Il faut néanmoins éviter cette permissivité car l'ensemble manque alors de cohérence❶.

2C. Propriété Chaîne vide autorisée

Autorise la saisie d'une chaîne de caractères de longueur 0. Cette propriété ne concerne que les champs de type *Texte* et *Mémo*. Attention à ne pas confondre l'absence de valeur (*Null*) et la valeur "" (chaîne vide) qui, bien que particulière, est quand même une valeur.

Pour distinguer les deux propriétés, lisez l'aide.

2D. Propriété Indexé

Un index facilite et accélère les requêtes, les recherches et les tris sur les tables. Access utilise l'index comme vous celui d'un livre : pour trouver un enregistrement d'après une valeur d'un champ indexé, il recherche la valeur de ce champ dans l'index et obtient la position de l'enregistrement dans la table. Il peut donc accéder directement dans la table à l'enregistrement voulu.

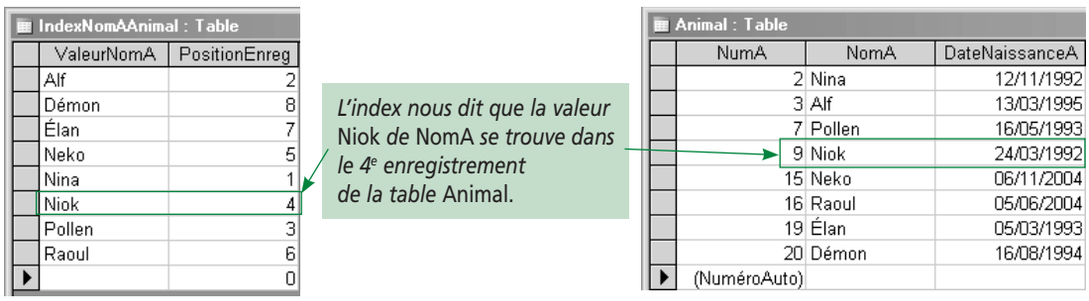
Si un champ n'est pas indexé, chercher les enregistrements ayant une valeur précise pour ce champ oblige à parcourir toute la table.

L'intérêt de l'index est double : il est toujours maintenu trié et ses enregistrements sont courts puisqu'ils contiennent uniquement le champ indexé et la position de l'enregistrement correspondant. Pour ces deux raisons, l'accès préalable à l'index est beaucoup plus rapide que le parcours de la table.

Par exemple, pour chercher la date de naissance de l'animal s'appelant Niok, on va s'aider du fait que le champ *Nom* est indexé. Ainsi, on va chercher dans l'index le numéro de l'enregistrement correspondant à l'animal s'appelant *Niok*. On peut alors accéder directement à l'enregistrement recherché dans *Animal* comme illustré ci-après

❶ *Dans la version Access 97, c'était une obligation : si vous autorisiez les valeurs nulles tout en définissant la propriété* Valide si*, vous deviez ajouter à cette dernière « Est Null ou … » pour expliciter que la valeur devait soit être nulle, soit vérifier une condition précise. Access 2000 a supprimé cette contrainte car elle passait mal, les non-informaticiens ne pensant pas à modifier* Valide si. *Cette permissivité est ennuyeuse pour nous car il faut bien reconnaître que la propriété* Valide si *n'est pas rigoureuse si on se permet de l'implicite.*

Voici la table *Animal* et l'index sur *NomA* créé❶ :



Remarque
Attention à la confusion ! Il faut bien distinguer *PositionEnreg* dans l'index (entre *1* et *n* si la table contient *n* enregistrements) et *NumA* dans *Animal*, identifiant dont la valeur est arbitraire. Par exemple, l'enregistrement lié à Niok est bien le 4e dans le tableau ; le fait que l'identifiant *NumA* vaille 9 est sans objet ici.

Le principe est simple :

- si je recherche le ou les animaux nés le 24/03/1992, Access va parcourir la table *Animal* du début à la fin puisqu'il ne dispose pas d'index ;
- en revanche, si je veux la date de naissance de Niok, Access va exploiter l'index en cherchant l'entrée correspondante. L'index étant maintenu trié, cela va très vite (voir la recherche dichotomique en cours de programmation). L'index indique que l'enregistrement cherché est le 4e. Access y accède directement et renvoie alors la valeur du champ *DateNaissanceA*, soit 24/03/1992.

Les index sont gérés de façon transparente par Access : lorsque vous pensez accéder directement à un enregistrement, vous passez en fait préalablement par l'index pour connaître sa position.
C'est le concepteur de la base qui décide quels sont les champs à indexer. Si l'index facilite l'accès à un champ, il ralentit les opérations de suppression, de modification et d'insertion : à chacun de ces traitements, les index doivent être reconstruits pour rester triés. Indexer tous les champs constitue donc une erreur de conception grave. En pratique, on définira un index sur un champ si l'on effectue fréquemment l'une des opérations suivantes :

- tri de la table sur ce champ ;
- recherche utilisant ce champ.

La clé primaire étant un champ privilégié pour rechercher des enregistrements, Access définit automatiquement un index dessus.

La propriété *indexé* peut avoir différentes valeurs :

Valeurs de la propriété *Indexé*	
Valeur	**Signification**
Non	Pas d'index sur ce champ
Oui - Avec doublons	Créer un index en autorisant des valeurs identiques (exemple : index sur un nom)
Oui – Sans doublons	Créer un index sur un champ dont toutes les valeurs sont uniques. *cas des clés primaires*

❶ *J'ai créé l'index comme une table Access pour illustrer mon propos. C'est bien entendu fictif. Notez le nom de l'index :* IndexNomAAnimal *pour « index sur le champ* NomA *dans la table* Animal ».

Je vous rappelle (nous l'avons vu dans le cours SQL) qu'un doublon est une répétition d'une valeur. La valeur de l'identifiant étant unique, il est normal que la clé primaire soit indexée sans doublon.
En revanche, si l'on définit un index sur une clé étrangère, il est logique d'accepter les doublons. Ainsi, dans les tables *Livre (*NumLivre*…, NumAuteur#)* et *Auteur (*NumAuteur*…)* :

- les clés primaires *NumLivre* et *NumAuteur* sont indexées sans doublons ;
- la clé étrangère *NumAuteur* dans *Livre* est indexée avec doublons, ceci matérialisant le fait que je peux posséder plusieurs livres du même auteur.

3. Les propriétés tournées vers l'utilisateur

J'ai déjà expliqué à plusieurs reprises l'importance d'une interface « intelligente », c'est-à-dire devançant les interrogations de l'utilisateur et empêchant au maximum les erreurs de saisie des données. Partant du principe qu'une application fonctionne, nous dirons qu'elle est :

- *médiocre* si elle plante lorsque l'utilisateur fait une saisie erronée ;
- *moyenne* si elle détecte l'erreur et affiche un message d'avertissement ;
- *bonne* si elle empêche l'erreur (par exemple, en interdisant de saisir des lettres dans un téléphone) ;
- ***excellente*** si l'interface est si bien conçue que l'utilisateur ne fait pas d'erreur car il est parfaitement guidé. En pratique, une faute de frappe est toujours possible ; une excellente application devra donc préalablement être bonne.

Les propriétés ci-dessous vous aideront à réaliser une excellente interface. Elles sont donc **indispensables**.

3A. Propriété Format

Le format correspond au **mode d'affichage** de la valeur ; il est indépendant de la valeur stockée. C'est exactement le même principe que le format appliqué aux cellules Excel. On choisira le format en fonction de la sémantique (et non du type) de la donnée. Par exemple, la valeur numérique 7365478821 sera affichée (formatée) :

- 73 65 47 88 21 si elle représente un numéro de téléphone ;
- 7 365 478 821 si c'est un nombre ;
- 7 365 478 821 € si c'est une valeur monétaire…

De même, les dates peuvent être affichées selon plusieurs formats : *20/05/1799* est le format abrégé, *20-déc-1799* le format réduit…❶.

Le format est très important dans l'ergonomie de l'application. En effet, l'utilisateur non-informaticien perçoit la donnée et son mode d'affichage comme un tout. Il aura bien du mal à reconnaître la date 07/12/2008 dans le texte 20081207 (format AAAAMMJJ). C'est un problème récurrent en informatique : l'utilisateur ne connaît de l'application que l'interface. L'affichage des données est donc crucial.

❶ *Access n'est pas étanche : il exploite certains réglages de Windows, notamment les* Paramètres régionaux *définis dans le* Panneau de configuration. *C'est là que vous indiquerez la façon française d'écrire la date, l'heure, quel symbole monétaire utiliser… la même base Access sur un ordinateur paramétré autrement donnera donc un affichage différent. Excel, Word et autres utilisent également ces réglages.*

Access propose de nombreux formats pour les dates, les chiffres… S'ils ne vous suffisent pas, vous pouvez définir les vôtres. Allez voir l'aide pour plus d'informations.

Changez quelques formats, par exemple celui de la date d'achat puis affichez le contenu de la table pour vérifier que les données sont affichées selon votre nouveau format.

3B. Propriété Masque de saisie

3B1. Définition

Le masque de saisie permet de définir les caractères autorisés et interdits lors de la saisie de la valeur d'un champ. Si vous entrez un caractère autorisé, il sera pris en compte. En revanche, la saisie d'un caractère interdit sera rejetée de façon douce : on attend des chiffres et vous appuyez sur une lettre ? Et bien, rien ne se passera, c'est comme si vous n'aviez rien tapé.
Le masque peut :

- limiter les caractères autorisés pour la saisie de la valeur du champ (exemple : un code postal ne doit être constitué que de chiffres donc appuyer sur les lettres ne produira rien) ;
- rendre facultatif ou obligatoire la saisie de certains éléments (exemple : la clé dans un numéro de compte) ;
- changer la casse des lettres saisies (exemple : mettre la première lettre d'un nom en majuscule). Cela permet d'uniformiser l'aspect des données.

Le masque rend plus intelligente la saisie des données et permet d'éviter beaucoup de fautes de frappe. Il apporte non seulement un gain de temps, mais aussi une fiabilité et une homogénéité accrues des données.

Concrètement, qu'est-ce qu'un masque ? C'est une chaîne de caractères. Et chaque caractère du masque contrôle le type d'un caractère que vous saisissez. Le 1er caractère du masque correspond au 1er que vous saisissez, le 2e au 2e… ❶

3B2. Le contenu du masque

En vrai, le masque est une chaîne de caractères constituée de une à trois sections séparées par des « ; ». Je ne détaille que la première, pour les deux autres, utilisez l'aide ou l'assistant.
Le tableau suivant présente la liste des principaux caractères pouvant former la première section du masque.

❶ *Enfin, pas tout à fait car le masque peut contenir des caractères de contrôle modifiant votre saisie comme > et < pour forcer le passage en majuscules ou minuscules. (Voir la suite du paragraphe.)*

Différents caractères constituant le masque		
Caractère	**Signification**	**La saisie est**
0	Chiffre (0 à 9), signes + et – refusés	obligatoire
9	Chiffre ou espace, signes + et – refusés	facultative
#	Chiffre ou espace, signes + et – acceptés	
L	Lettre (A à Z, a à z)	obligatoire
?		facultative
A	Lettre ou chiffre	obligatoire
a		facultative
&	Caractère quelconque ou espace	obligatoire
C		facultative
\	Le caractère qui **suit** dans le **masque** est affiché tel quel (il ne contrôle donc pas la saisie). Un caractère sans ambiguïté (donc pas 0, 9, #, L, ?, A, a, &, C, \, < et >) peut se passer d'être préfixé par \	*sans objet (caractères de contrôle)*
<	Convertit tous les caractères saisis qui suivent en majuscules (toute saisie, minuscule ou majuscule, s'affiche donc en majuscule)	
>	Convertit les caractères qui suivent en minuscules	

Notez que vous pouvez filtrer trois familles de touches : les chiffres, les lettres et le caractère espace.

Pour bien comprendre, envisageons le masque `09>LL`. Quelles données va-t-il permettre de saisir ?

- le 1er caractère du masque est *0*. Le premier caractère saisi doit donc obligatoirement être un chiffre ;
- le 2e caractère du masque est *9*. Le deuxième caractère peut donc être un chiffre ou être sauté (on passe donc au caractère suivant dans le masque) ;
- le 3e caractère est >. Il ne correspond à aucune saisie de l'utilisateur mais indique à l'application que toutes les lettres qui vont maintenant être saisies doivent être converties en majuscules ;
- le 4e caractère du masque est *L*. Le troisième❶ caractère saisi doit donc être une lettre ;
- idem pour le 5e caractère (qui est le 4e si le second chiffre n'a pas été saisi).

Finalement, la donnée saisie doit être constituée d'un ou deux chiffres suivis par deux lettres. Les lettres seront en majuscules (donc, si elles sont saisies en minuscules, l'application les convertira en majuscules). La donnée saisie fera donc 3 ou 4 caractères. Voici deux exemples de données acceptées par le masque :

- 8AB ;
- 12ZE.

Il faut bien observer que le masque fonctionne de façon purement syntaxique. Au fur et à mesure de votre saisie, les touches interdites sont désactivées. C'est tout. Le masque ne permet pas de vérifier la valeur de la saisie. Si grâce à lui on peut obliger à saisir un chiffre, on ne peut imposer qu'il soit compris entre 5 et 9.

❶ *Ou le deuxième si le second chiffre, facultatif, n'a pas été saisi… La notion de saisie facultative n'est pas si évidente !*

Remarques :

- un masque de saisie dispose encore d'autres fonctionnalités… voir l'aide ; vous pouvez notamment présenter un gabarit à l'utilisateur. Par exemple, une zone de texte où l'on doit saisir une date contiendra « __/__/____ ». C'est très lisible !
- pour un contrôle vraiment poussé de la saisie (p. ex. un numéro de sécurité sociale commence par le chiffre *1* ou *2*), le masque n'est plus suffisant. Il faut utiliser la propriété *Valide si* (voir le paragraphe 3E) ou travailler par programme ;
- l'aide en ligne vous conseille avec raison d'utiliser l'assistant *masque de saisie* pour définir vos masques.

Le rôle du masque est assez évident mais son codage par une chaîne de caractères est tarabiscoté, d'autant que l'on mélange le filtrage des caractères avec leur mise en forme (majuscule ou minuscule). La gestion des caractères facultatifs augmente la complexité. C'est ennuyeux car le masque est une fonctionnalité essentielle dont vous ne **devez pas** vous passer. Faites des tests et consultez l'aide pour être sûr de maîtriser cet outil.

3C. Propriété Légende

La légende sera utilisée comme libellé des contrôles associés aux champs dans les formulaires et les états. (Nous étudierons cela plus tard.) Sans légende, c'est le nom du champ qui est utilisé.
La légende apparaîtra au bas du formulaire lors de la saisie du champ correspondant. C'est une aide augmentant l'ergonomie de l'application. Pour une date de naissance, on mettra par exemple « date de naissance : jj/mm/aaaa ». L'utilisateur saura alors quoi saisir et comment le saisir. Bien entendu, vous exploiterez en même temps le masque !

3D. Propriété Valeur par défaut

Si un champ prend très souvent la même valeur, il est intéressant de l'initialiser par défaut à cette valeur. L'utilisateur n'aura donc à saisir le champ que lorsqu'une valeur différente devra être saisie.
Par exemple, Access place *0* par défaut pour les champs numériques et monétaires.

Il est également possible de placer des expressions comme valeur par défaut. Ainsi, on peut initialiser un champ *date* à la date du jour par l'expression =Date(). On retrouve la syntaxe Excel : le « = » indique une formule, *Date* est une fonction ne prenant aucun paramètre (d'où les parenthèses vides) et renvoyant la date système.

3E. Propriété Valide si

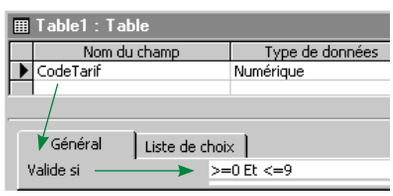
Le masque de saisie définit la syntaxe de la saisie en obligeant à entrer les bons caractères (lettre, chiffre ou espace pour chaque touche appuyée). On peut aller plus loin en contrôlant la valeur saisie. Par exemple, il est bon de s'assurer qu'une date de facture ou de règlement n'est pas postérieure à la date du jour. Pour faire cela, on utilisera une règle de validation.

La règle est exprimée sous la forme d'une expression booléenne. Les opérateurs habituels de comparaison (<, >, =…) et les connecteurs booléens (*et*, *ou* et *non*❶) sont autorisés. La saisie ne sera acceptée que si la règle de validité (c-à-d. la condition) est vérifiée.
Attention, la syntaxe est un peu tordue pour un informaticien.

Supposons que l'on veuille saisir un code tarif compris entre 0 et 9 inclus. Si le champ s'appelle *CodeTarif*, il devra trivialement vérifier la condition *CodeTarif>=0 et CodeTarif<=9*. Cette notation est limpide pour nous, les informaticiens. En revanche, le néophyte ne sait pas écrire de telles choses (soyez honnête et rappelez-vous vos débuts !). Fidèle à sa politique d'accès au plus grand nombre (le marché de l'humain est plus vaste que celui de l'informaticien), Microsoft a choisi une notation collant à la langue parlée❷. L'idée est simple : on écrira la condition sans faire référence au champ lui-même. Nous supprimons donc le nom *CodeTarif*.

On obtient alors la règle de validation >=0 et <=9.
Je disais que c'est tordu pour un informaticien car cette syntaxe est erronée en programmation. En revanche, que pensera un néophyte en voyant ce qui suit ?

Il lira comme c'est écrit (les flèches indiquent le sens de lecture), à savoir :
« *CodeTarif* est valide s'il est supérieur ou égal à 0 et inférieur ou égal à 9 ».



Du coup, c'est du français tout simple. Réciproquement, une fois la phrase en français écrite, le débutant saura définir la valeur de *Valide si*.
Pour une fois, les rôles sont renversés : comme c'est le point de vue du non-spécialiste qui a été retenu, c'est le spécialiste qui ne comprend pas trop la syntaxe. Aux informaticiens mécontents de ce choix, on rétorquera qu'il vaut mieux demander au spécialiste de s'adapter car, par définition, cela lui est plus facile qu'au débutant.
Savoir si Microsoft a eu raison ou tort est une autre histoire car la question sous-jacente est toujours de savoir si un SGBD est ou non accessible à un non-informaticien.

Après cette digression des plus intéressantes, revenons à l'écriture de notre propriété *Valide si*. Les dates doivent être mises entre # et le texte entre guillemets (comme sous SQL). Vous bénéficiez de l'opérateur *Comme* (c'est le *like* de SQL) en lien avec les caractères génériques *?* (qui remplace un caractère) et * (qui remplace plusieurs caractères).

Voici quelques exemples ; l'aide en contient d'autres. Retenez que la valeur saisie reste implicite dans l'expression booléenne.

❶ *Attention, il faut vraiment mettre les connecteurs en français.*

❷ *En fait, comme l'utilisateur non-averti fait toujours la même erreur, Microsoft a validé cette erreur en en faisant la règle d'écriture de la propriété* Valide si. *C'est un peu l'idée du proverbe disant qu'il vaut mieux avoir tort avec les autres que raison tout seul. Dit autrement, il peut être préférable de légitimer une erreur commune plutôt que de tenter à toute force d'éduquer l'utilisateur. Vous remarquerez la subtilité de cette approche : Microsoft n'a rien laissé au hasard ! Cette propriété est un modèle d'ergonomie (nous étudierons cela dans la séquence 13… rassurez-vous, treizième et dernière !).*

Règle de validité	Explication
<>0	la valeur saisie, numérique, ne doit pas être nulle
<>"0"	la valeur saisie (du texte) ne doit pas être le *caractère 0* (la saisie *00* sera acceptée)
>50	la valeur doit être supérieure à 50
Comme "A?????"	la saisie doit commencer par *A* et faire 6 caractères
>#12/12/2006#	la date saisie doit être postérieure au 12/12/2006
<=Date()	la date saisie doit être antérieure ou égale à la date du jour
(>=0 Et <10) Ou =20	la valeur saisie doit être égale à 20 ou comprise entre 0 et 9

Si vous modifiez la règle de validité d'un champ dans une table déjà remplie, Access va s'assurer que les valeurs existantes vérifient la nouvelle condition. Vous serez prévenu en cas de problème…

3E1. Exercice d'application : sujet

Je vous rappelle qu'un numéro de sécurité sociale est constitué de treize chiffres (version courte) ou de quinze (version longue). En réalité, seuls les treize premiers chiffres forment le numéro proprement dit. Lorsque vous donnez un numéro à quinze chiffres, vous fournissez en fait le numéro sur treize chiffres et la clé facultative codée sur deux chiffres (13+2 = 15). La clé permet de vérifier qu'il n'y a pas d'erreur dans le numéro (cela permet d'éviter les fautes de frappe).

Exercice

Créez une nouvelle table. Ajoutez-lui un champ *NumSécu* qui devra faire treize ou quinze chiffres. **Définissez toutes les propriétés,** notamment le masque de saisie et la règle de validité. Testez votre travail en saisissant des données dans la table.

Notez que le masque du numéro de sécurité sociale est proposé par l'assistant. Pour une fois, ne l'utilisez pas !

3E2. Exercice d'application : corrigé

Le masque de saisie

On veut 15 chiffres, dont 13 obligatoires et 2 facultatifs. Cela donne 000000000000099. Comme on peut définir le gabarit avec le masque, on va séparer la saisie en groupe de chiffres conventionnels. En effet, un numéro de sécurité sociale s'écrira ainsi : 2 10 06 59 665 666 88. Il nous suffit de rajouter des espaces dans le masque. Cela donne :

0 00 00 00 000 000 99

(Access rajoute le caractère de contrôle « \ » devant chaque espace.)

Notez que ce n'est pas parfait car la clé, c'est *deux chiffres sinon rien*. Il faudrait donc pouvoir rentrer 13 ou 15 chiffres, mais pas 14. Cela n'est pas à la portée du masque.

La règle de validité

On va vérifier que le numéro commence par un *1* ou un *2*. Notez que cette donnée doit être définie en texte comme on l'a vu dans le paragraphe 2C de la séquence 2. On n'utilisera donc pas de comparateur arithmétique mais l'opérateur *Comme*.
Autre problème : le champ pouvant avoir 13 ou 15 chiffres, on devrait dire : treize ou quinze caractères (forcément des chiffres d'après le masque) commençant par *1* ou *2*.

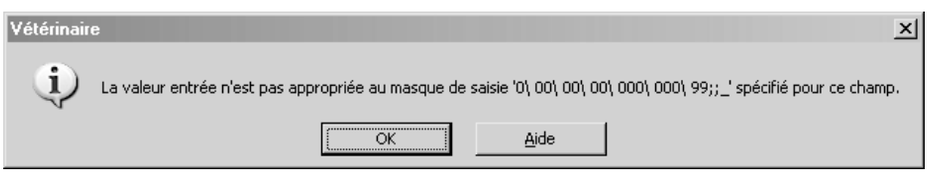
Cela donne quatre combinaisons, soit :

Comme "1?????????????" ou Comme "2?????????????" ou Comme "1???????????????"
ou Comme "2???????????????"

N'oubliez pas de définir la valeur de *Message si erreur*.

Comme la règle *Valide si* ci-dessus n'est pas lisible ni aisée à utiliser, on peut exploiter le fait que la longueur du numéro (13 ou 15 chiffres) est déterminée par le masque. Dans la règle de validité, on peut se contenter de vérifier que le numéro commence par *1* ou *2*, quelle que soit la suite de chiffres.
On écrira donc : Comme "1*" ou Comme "2*"

Le problème, c'est que *Message si erreur* ne peut plus tenir compte de la longueur erronée. Si vous tapez le numéro 1234, vous obtiendrez un message d'erreur standard d'Access basé sur le non-respect du masque. (Il n'est pas possible d'en définir un soi-même pour ce cas.)
Voici le message en question ci-dessous. Imaginez la tête de l'utilisateur !



Complément

La clé d'un numéro de sécurité sociale (les deux derniers chiffres optionnels) permet de vérifier la validité d'un numéro. Cela évite toute erreur de saisie. Pour l'exploiter, il faut faire un calcul arithmétique sur les chiffres. C'est hors de portée de notre règle de validité. Si nous voulons faire ce calcul, il faudra le faire par programmation dans les formulaires.

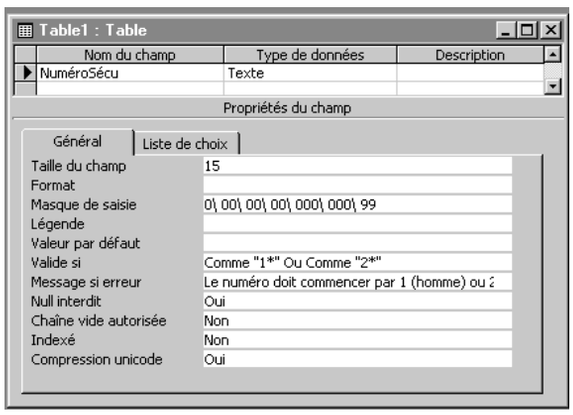
Une petite remarque

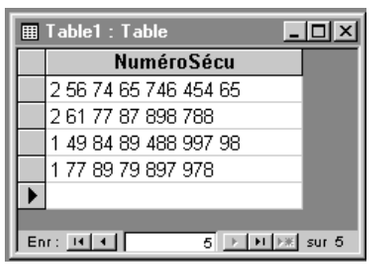
Si vous avez testé cela sous Access (ce que vous avez fait si vous êtes sérieux), vous aurez observé qu'Access a mis un index sur *NumSécu* (je l'ai enlevé ci-dessous). Pourquoi ? C'est le côté parfois pénible *je m'occupe de tout* des outils Microsoft : allez voir le menu *Outils/Options*, onglet *Tables/Requêtes*, zone *Index automatique*.

Autres propriétés

Il fallait penser à définir la taille du champ à 15 caractères, soit la plus grande longueur possible. J'ai également défini *Message si erreur* (le texte complet est *Le numéro doit commencer par 1 (homme) ou 2 (femme)*) et les propriétés qui suivent. Voyez les paragraphes suivants pour l'explication de ces propriétés.

Les copies d'écran suivantes vous présentent la définition du champ et ses propriétés ainsi que les exemples de saisie des différents cas. Testez cela et vérifiez que vous ne pouvez pas saisir de numéro invalide.



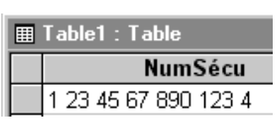


Remettons en cause notre travail

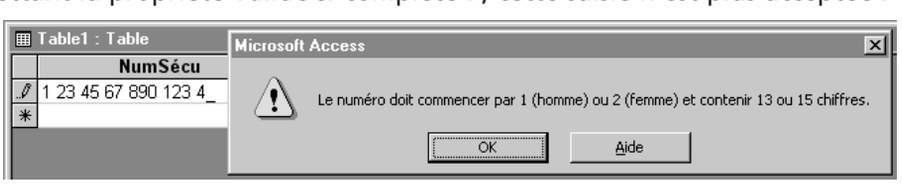
Lors de l'écriture de ce cours, j'avais écrit la propriété *Valide si* avec les quatre termes *Comme*. Lors de la première relecture, j'ai eu l'idée de simplifier l'expression en passant à seulement deux termes. J'avoue que j'étais fier de moi… La deuxième relecture m'a permis de me rendre compte du problème d'indivisibilité de la clé (elle est soit présente sur deux chiffres, soit absente. Le numéro de sécurité sociale doit donc faire treize ou quinze chiffres mais pas quatorze). C'est à la suite de la troisième relecture (et oui, la troisième) que j'ai rajouté cette partie *remise en cause*.

En fait, la troisième relecture m'a fait prendre conscience que ma simplification de *Valide si* était hâtive. En effet, la version avec les quatre *Comme* permettait d'imposer un nombre à treize ou quinze chiffres, ce qui n'est pas redondant avec le masque qui lui autorisait treize, quatorze ou quinze chiffres.

Voyons cela ; avec mon *Valide si* simplifié, je peux rentrer un nombre à quatorze chiffres donc fatalement erroné :



En remettant la propriété *Valide si* complète❶, cette saisie n'est plus acceptée :



Notez au passage que j'ai modifié la propriété *Message si erreur* qui contient maintenant *Le numéro doit commencer par 1 (homme) ou 2 (femme) et contenir 13 ou 15 chiffres.*

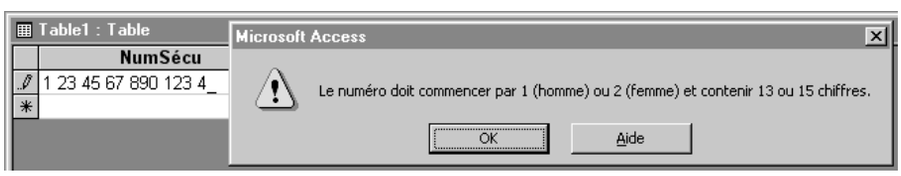
En guise de conclusion, je voudrais critiquer (de façon constructive) ce cours : pourquoi ai-je pris la peine de vous signaler ce qu'il faut bien appeler une erreur de ma part ? Il aurait été naturel de ne rien dire de mes relectures et de corriger directement mon support comme je l'ai fait jusqu'à présent. Si j'en ai parlé, c'est à des fins pédagogiques : l'exercice semblait assez simple mais finalement ne l'était pas.

❶ Comme "1?????????????" ou Comme "2?????????????" ou Comme "1??????????????" ou Comme "2??????????????"

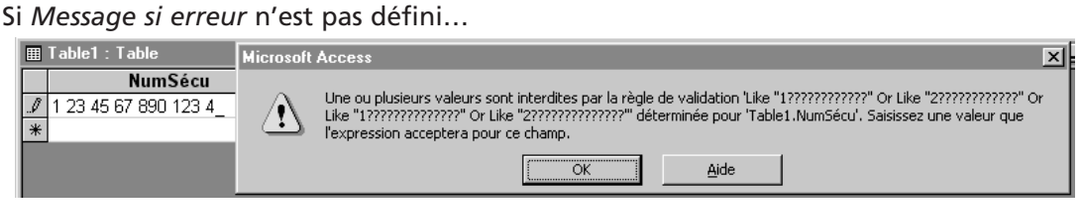
La preuve, j'ai fait une erreur. La leçon est de ne pas traiter toutes ces propriétés en ayant la tête ailleurs : ce n'est pas très difficile, mais il faut tout de même être attentif… d'autant que mon erreur n'était pas anodine puisqu'elle permettait la saisie de données erronées, ce qui est réellement **gravissime**.

3F. *Propriété Message si erreur*

Si la saisie viole la règle de validité, un message abscons mentionnant l'expression booléenne de validité s'affiche. Ceci n'est évidemment pas convivial ! La propriété *Message si erreur* permet de définir le message à afficher dans ce cas.
Pour vous le prouver, reprenons l'exercice précédent avec le numéro de sécurité sociale. Je vais de nouveau saisir un numéro à quatorze chiffres, avec puis sans texte dans *Message si erreur*. Je vous invite à tester cela par vous-même.
Si une valeur est définie pour *Message si erreur*, elle est utilisée :



Si *Message si erreur* n'est pas défini…



Dois-je conclure ? Et bien, il n'est pas réaliste d'espérer que l'utilisateur comprenne ce dernier message.

Pour servir à quelque chose, le message devra reprendre, en français, la sémantique de la règle de validité. Pour la saisie du numéro de sécurité sociale, j'ai indiqué et expliqué les valeurs permises. Il est important d'expliquer quand c'est possible. Cela fait moins « baguette magique » et l'utilisateur retient mieux ce qu'on lui explique, de même que vous retenez mieux des règles expliquées et justifiées plutôt qu'imposées.

Une dernière chose qui a son importance : à moins d'une grosse erreur de ma part, il est impossible de définir un message d'erreur s'appliquant au masque (la propriété *Message si erreur* ne traite que les erreurs liées à *Valide si*). Ainsi, lorsque je tente de ne rentrer que huit chiffres pour mon numéro de sécurité sociale, c'est le masque qui rejette la saisie puisqu'il oblige à rentrer entre treize et quinze chiffres comme l'on l'avons vu précédemment. J'obtiens alors :

Pour le coup, on obtient un message qui n'est pas des plus limpides. Je ne vois aucune solution. Cela dit, les tirets présents dans la zone de saisie et correspondant aux chiffres oubliés sont assez parlants.
Profitons de cette dernière remarque pour observer que c'est uniquement lorsque le masque valide la saisie (cette dernière est donc acceptable) que *Valide si* est évalué pour déterminer si la saisie peut finalement être acceptée.

4. Conclusion sur les propriétés

Imaginez-vous dans le meilleur des mondes où personne ne fait d'erreur. Toutes ces fonctionnalités de vérification sont alors inutiles ! Mais, dans la vraie vie, tout le monde peut faire des erreurs en saisissant des données. Cela va de la faute de frappe bête et méchante à l'erreur venant de l'incompréhension de la donnée attendue.
Une erreur est dramatique : de même que l'on peut démontrer n'importe quoi avec des hypothèses fausses, la base de données ne sera pas fiable si les données entrées sont invalides. Il convient de vérifier autant que possible la validité des données entrées :

- un âge est compris entre 0 et 120 ;
- un code postal est constitué de 5 chiffres ;
- une date de facture ne doit pas être postérieure à la date du jour ;
- …

Vous remarquerez que ces vérifications sont à la fois syntaxiques (vérification des **caractères** constituant la donnée) et sémantiques (vérification de la **valeur** de la donnée).
Bien entendu, l'application ne peut pas vérifier parfaitement les données : si j'entre dans la base que j'ai 90 ans, la saisie sera validée car elle est plausible. De même, si vous saisissez 16/08/2008 au lieu de 17/08/2008 pour une date de naissance, l'application ne pourra pas détecter l'erreur.

On se contente donc de faire au mieux. Nos deux objectifs sont :

- le confort de saisie (l'ergonomie) ;
- la qualité de la saisie (la validité des données).

Notons que vous pourriez faire tout cela par programmation bien plus tard lors de l'écriture de l'application. Mais alors, quel est l'intérêt de définir toutes ces propriétés lors de la création des champs ?
Cela va vous simplifier énormément le travail : dans tout formulaire ou état utilisant ces champs, tout ce que vous avez défini (masque, format…) sera automatiquement utilisé. Si vous ne fournissez pas ces informations lors de la définition des champs, il faudra tout refaire à chaque utilisation du champ dans un formulaire ou un état.

Synthèse

Que retenir de cette séquence ? Et bien, d'assez nombreuses choses !

Une table est constituée de champs qu'il faut décrire très précisément par les nombreuses propriétés permettant de les définir finement (sous-type du champ, ergonomie et contrôle de la saisie).

J'ai beaucoup insisté sur l'importance de la définition fine et précise des champs. En effet, c'est un peu fastidieux mais c'est crucial :

- pour la suite du développement de l'application car cela vous simplifiera le travail lors de la création des formulaires et des états ;
- pour le fonctionnement futur de l'application (rejet des données erronées).

N'oubliez pas que notre exercice sur le numéro de sécurité sociale nous a entraînés beaucoup plus loin que je ne l'avais initialement prévu. Cela montre que ces propriétés sont plus délicates qu'il n'y paraît.

Cela méritait bien dix-sept pages denses mais intéressantes, non ?

Séquence 4

Relations et intégrité référentielle

Dans cette séquence, nous terminons la définition des tables en les reliant entre elles pour identifier les clés étrangères. Nous étudierons également finement un outil très performant : l'intégrité référentielle.

▶ *Capacités attendues*

- **Savoir créer les relations**
- **Comprendre le rôle de l'intégrité référentielle et pouvoir l'appliquer à bon escient**

▶ *Contenu*

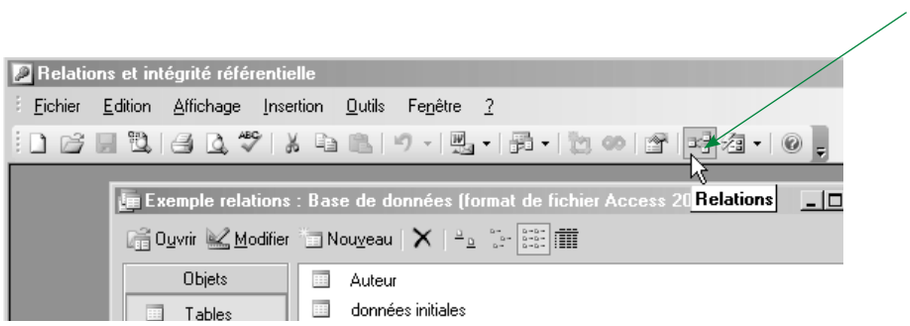
1. Relations entre les tables

1A. Définition

L'association [1–n]❶ du MCD est représentée par la notion de clé étrangère dans le MLD. Comment traduire cela sous Access ? Nous avons vu dans le paragraphe 2D de la séquence 2 que cela ne se faisait pas lors de la définition des tables. Le fait de donner le même nom aux deux champs du couple *clé primaire/clé étrangère* n'y change rien.
Vous devez réaliser une opération spécifique permettant à Access d'identifier les clés étrangères associées aux différentes clés primaires. Cela va se faire en reliant les clés primaires à leurs clés étrangères par le biais de *relations*. Les relations permettent également de définir l'intégrité référentielle.
Attention, les relations sont souvent négligées par les étudiants. Si vous ne les mettez pas en place, Access n'a aucun moyen de relier les tables entre elles. Avant d'aller plus loin, consultez l'aide d'Access pour une première initiation : cherchez *relation tables* puis choisissez la rubrique *Relations entre les tables* puis *À propos des relations dans une base de données Access (MDB)*.

1B. Mise en œuvre

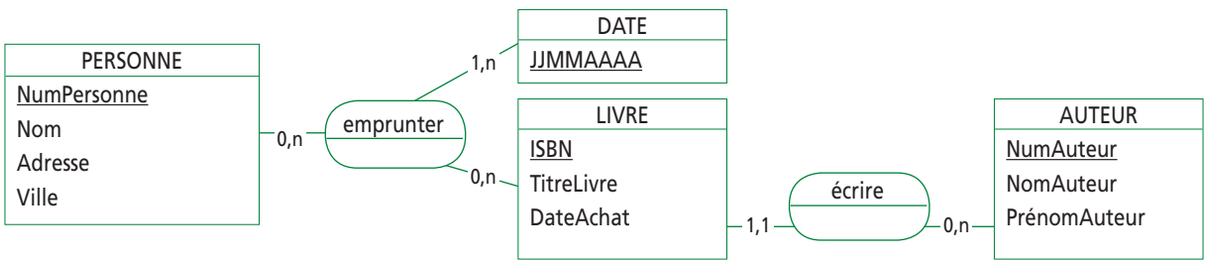
Les relations concernent l'ensemble de la base et non une table ou un formulaire. Il faut donc que la fenêtre *base de données* soit active (cliquez dessus ou faites *F11*) pour y accéder❷.
Lorsque c'est le cas, lancez la commande *Outils/Relations…* ou cliquez sur le **bouton** correspondant dans la barre d'outils *Base de données*.



Établir des relations n'a de sens que si vous avez des champs à relier, c'est-à-dire au moins un couple *clé primaire/clé étrangère*. Lisez ce qui suit avant de passer à la réalisation.
La première étape consiste à ajouter les tables participant aux relations, donc toutes les tables possédant un élément de couple *clé primaire/clé étrangère*. Normalement, toutes les tables de la base sont concernées puisqu'elles sont reliées par au moins une association.
La seconde étape consiste à relier chaque clé primaire avec sa clé étrangère. Cela se fait graphiquement à la souris.

❶ Je vous rappelle que la notation [1-n] indique une binaire ayant en cardinalités maximales 1 pour une patte et n pour l'autre. (Relisez le paragraphe 4D de la séquence 3 du cours d'analyse.)
❷ Ouvrez une table quelconque et réduisez la taille de la fenêtre pour voir à l'écran la fenêtre de la base et celle de la table. Cliquez alors alternativement dans l'une puis dans l'autre. Observez que les menus et les barres d'outils changent en fonction de l'objet sélectionné : Access ne vous propose que les commandes pertinentes associées à l'objet en cours de manipulation.

Je vais illustrer la démarche sur un petit exemple. Voici le MCD :



Le MLD est alors :

Personne (NumPersonne, Nom, Adresse, Ville)
Livre (ISBN, TitreLivre, DateAchat, NumAuteur#)
Emprunter (NumPersonne#, JJMMAAAA, ISBN#)
Auteur (NumAuteur, NomAuteur, PrénomAuteur)

primaire, étrangère#

(Si vous ne comprenez pas pourquoi la table Date *n'est pas créée, allez voir le cours d'analyse, précisément séquence 5, paragraphe 2H1.)*

Nous avons les couples *clé primaire/clé étrangère* suivants :

- *NumAuteur* (clé primaire dans *Auteur*, clé étrangère dans *Livre*) ;
- *NumPersonne* (primaire dans *Personne*, étrangère dans *Emprunter*) ;
- *ISBN* (primaire dans *Livre*, étrangère dans *Emprunter*).

Première étape, je crée les tables comme nous avons appris à le faire.

Nous lançons ensuite l'outil *Relations*. La fenêtre des relations est naturellement vide. Dans un premier temps, nous ajoutons les tables concernées par les relations. Vous remarquerez que les clés primaires sont en gras.

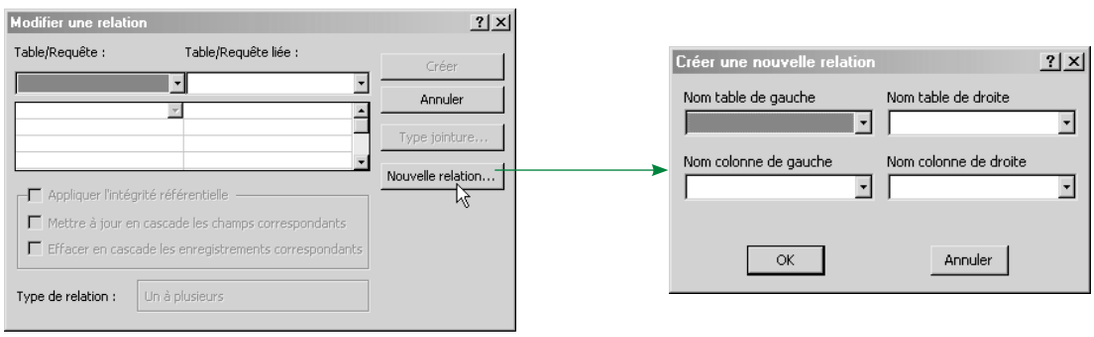
Comment créer une relation ? Je vous rappelle que l'objectif est de faire le lien entre la clé primaire d'une table et les clés étrangères correspondantes situées dans d'autres tables❶. Il faut donc associer un champ d'une table à un champ d'une autre table. Il y a deux techniques.

1B1. Première technique

Nous allons créer la relation sur le champ *NumAuteur* entre les champs *NumAuteur* des tables *Auteur* et *Livre*.

❶ *Une clé primaire sera associée à plusieurs clés étrangères si l'entité participe à plusieurs associations.*

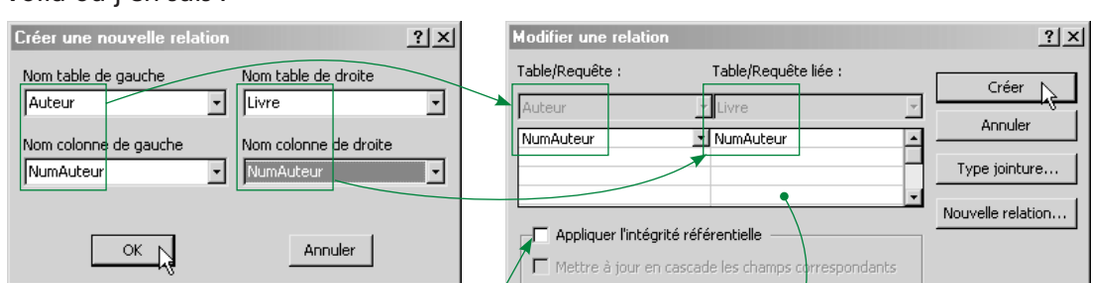
Il faut lancer le menu *Relations/Modifier une relation…* puis, dans la boîte de dialogue *Modifier une relation*, cliquer sur le bouton *Nouvelle relation…* :



Vous devez alors indiquer les tables et les champs. Access vous propose des listes déroulantes. Il faut :

- d'abord choisir la table dans la liste. Access propose les différentes tables ajoutées dans la fenêtre *Relations* ;
- ensuite, choisir la colonne associée (ce qu'Access appelle *colonne*, nous l'appelons *champ*). Access propose les différents champs de la table sélectionnée précédemment.

Attention à l'ordre des tables ! C'est le même dans les deux boîtes de dialogue, même si c'est plus explicite dans *Modifier une relation* que dans *Créer une nouvelle relation* : **la clé primaire et sa table se mettent à gauche, la clé étrangère et sa table à droite**❶.
Voilà où j'en suis :



C'est quoi tout ça ? Utilisez l'aide contextuelle de la fenêtre : cliquez sur le bouton ? dans la barre de titre puis sur l'élément pour lequel des informations seraient les bienvenues.
L'intégrité référentielle sera étudiée dans la suite de la séquence.

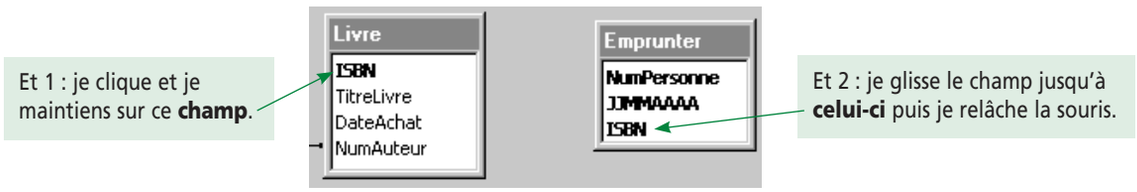
Ces listes *permettent d'ajouter les autres champs dans le cas de clés multichamps.*

Ne mettez pas d'intégrité référentielle, nous verrons cela dans le paragraphe suivant. Validez par *Créer…* la relation est constituée. Elle est matérialisée par un trait entre les deux tables *Auteur* et *Livre*. Placez le curseur souris juste sur le trait, double-cliquez… vous retrouvez la fenêtre ci-dessus, ce qui vous permet de modifier la relation.

❶ *Les notions de « table de gauche » et « table de droite » n'ont donc aucun rapport avec les positions respectives des tables dans la fenêtre graphique des relations.*
Dans le cas d'une clé monochamp, Access distingue la clé primaire et la mettra spontanément à gauche. Il ne saura en revanche pas le faire si la clé est multichamps. Le respect de l'ordre indiqué dans le cours est alors indispensable.

1B2. Seconde technique

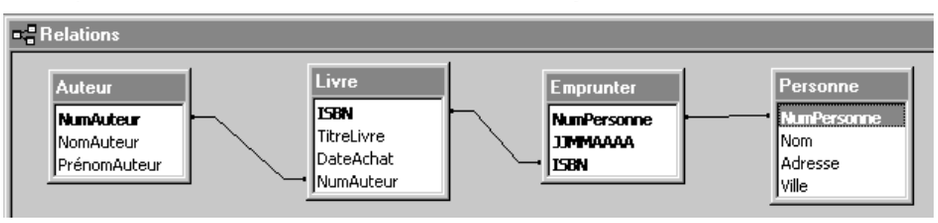
Je vous la conseille, elle est plus rapide ! Nous allons créer la relation sur *ISBN* entre *Livre* et *Emprunter*. Vous cliquez et laissez appuyé sur un des champs (clé primaire ou étrangère) et vous le glissez sur l'autre champ. Vous lâchez… la relation est créée. Visuellement :



Quelle que soit la technique utilisée, les champs reliés doivent être de même type vu la sémantique du lien : la relation indique que les valeurs des champs seront comparées. De même, les champs de type *Numérique* doivent comporter la même définition pour la propriété *Taille de Champ*. (Rappel : clé primaire *NuméroAuto* → clé étrangère *Numérique/entier long*.)

Lorsque toutes vos relations sont définies, fermez la fenêtre *Relations* en prenant soin d'enregistrer votre travail. Les relations seront utilisées automatiquement lors de l'écriture des requêtes et la conception de formulaires ou d'états.

Pour modifier les relations ou en ajouter… il suffit de rouvrir la fenêtre.
Je continue pour créer toutes les relations. Au final, j'obtiens la fenêtre suivante :



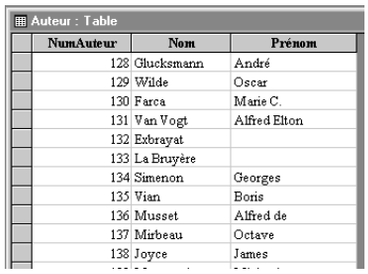
Les couples *clé primaire/clé étrangère* sont formés par *NumAuteur* dans *Auteur* et *Livre*, *ISBN* dans *Livre* et *Emprunter* et *NumPersonne* dans *Emprunter* et *Personne*.

1C. Visualisation des relations dans les tables

En reliant clés primaires et étrangères, on relie en fait les enregistrements sous-jacents. Par exemple, la relation sur *NumAuteur* entre *Livre* et *Auteur* permet d'obtenir tous les livres d'un auteur donné. Pour avoir les livres écrits par Oscar Wilde (valeur de clé primaire *NumAuteur* 129), je cherche la valeur 129 pour la clé étrangère *NumAuteur* dans *Livre*. J'obtiens alors tous les livres de Wilde.

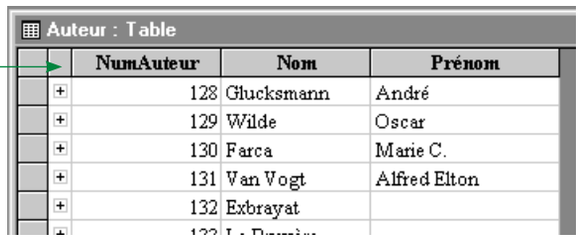
Comprenez bien qu'une fois la relation définie, ce que l'on vient de voir est automatique : une valeur de clé primaire correspond à 0 ou 1 ou plusieurs enregistrements ayant même valeur pour la clé étrangère. On peut le visualiser à partir des tables. Pour illustrer cela, je vais dans un premier temps supprimer toutes les relations dans la base… Voilà, c'est fait. Le contenu d'*Auteur* est présenté ci-dessous.

J'ai bien réduit la fenêtre car il n'y a rien de spécial à voir.



Je crée maintenant la relation entre les champs *NumAuteur* de *Livre* et *Auteur*. En ouvrant de nouveau *Auteur*, la fenêtre change :

Qu'y a-t-il de neuf ? Cette **colonne** contenant des « + ». Ce sont les mêmes « + » que dans l'explorateur : cliquer dessus permet de développer la ligne.

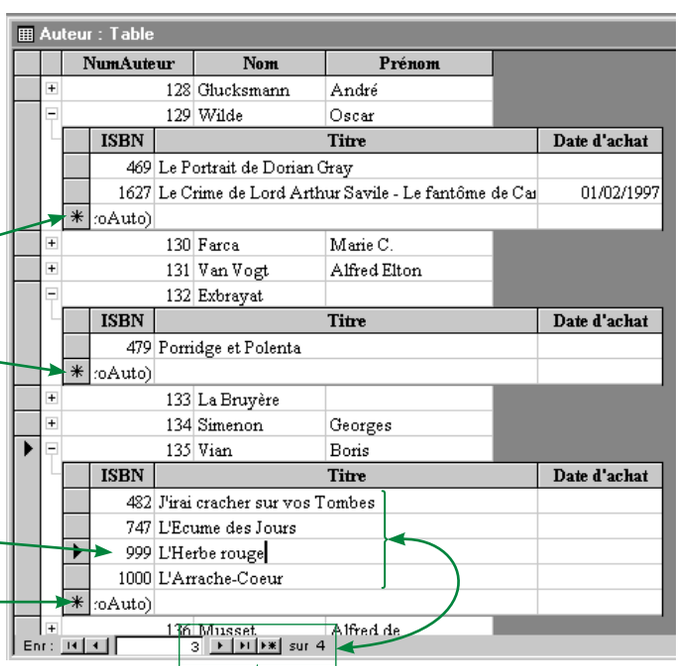


Sous l'explorateur, cliquer sur le « + » devant un dossier le développe et affiche les sous-dossiers qu'il contient.

Ici, cliquer sur le « + » exploite la relation en donnant les livres associés à l'auteur. J'ai souhaité avoir la liste des livres de Wilde, Exbrayat et Vian. J'ai donc cliqué sur leur « + » et j'obtiens :

Je pourrais rentrer un nouveau livre de Wilde (ou Exbrayat ou Vian) en remplissant un de ces enregistrements *. L'enregistrement correspondant sera créé dans *Livre*. Quelle sera la valeur de *NumAuteur* pour le livre ? Et bien, celle de l'auteur à qui je rajoute un livre !

Je suis actuellement sur cet enregistrement de la table *Livre*. Si je le modifie, il sera mis à jour dans *Livre*.



Vous remarquez que je suis bien dans le sous-ensemble des livres de Vian (je suis sur le 3e livre, sachant que j'en possède 4).

En créant les autres relations, je disposerai des mêmes fonctionnalités : les relations entre *Emprunter* et *Livre* d'une part et *Emprunter* et *Personne* d'autre part me permettent d'avoir les livres empruntés par chaque personne à partir de la table *Personne*. Symétriquement, j'ai les emprunts de chaque livre à partir de la table *Livre* (attention, je dis bien les emprunts et non les emprunteurs). Mais au fait, j'ai également accès à *Livre* par *Auteur* ! Ainsi, je peux maintenant connaître l'emprunteur d'un livre d'un auteur donné comme le montre la copie d'écran présentée ci-dessous.

Vous remarquerez la colonne des « + » dans la liste des livres. Elle vient de la relation avec *Emprunter*.

Les raisins de la colère a été emprunté par les personnes 2 et 10.

À l'est d'Eden n'a jamais été emprunté.

Auteur : Table

NumAuteur	Nom	Prénom
40	James	Henry
41	Perec	Georges
42	Steinbeck	John

ISBN	Titre	Date d'achat
139	La grande Vallée	
140	Rue de la Sardine	
141	Tendre jeudi	
508	En un Combat douteux	
656	Des Souris et des Hommes	
657	La Perle	
658	Les Raisins de la Colère	

NumPersonne	JJMMAAAA
2	04/03/2001
10	13/04/1999
0	

ISBN	Titre	Date d'achat
709	Tortilla Flat	
774	A l'Est d'Eden	

NumPersonne	JJMMAAAA
0	

ISBN	Titre	Date d'achat
877	Les Pâturages du Ciel	
931	Les Naufragés de l'Autocar	
1226	Au Dieu inconnu	01/10/1994
1325	Une saison amère	21/04/1995
1496	Le Règne éphémère de Pépin IV	30/03/1996
1674	Voyage avec Charley	19/07/1997
1705	La Coupe d'or	11/06/1998
:oAuto)		

Je viens de dire que *À l'Est d'Eden* n'a pas été emprunté. On a pourtant l'impression que la personne numéro 0 l'a emprunté à une date indéterminée. Qui a raison ? Moi, bien sûr ! Mais quand ai-je raison ? Quand je dis que le livre a été emprunté ou non ?
Vous remarquerez que le seul enregistrement d'*Emprunter* affiché pour ce livre est l'enregistrement *. Or, cet enregistrement n'en est pas un : c'est juste un gabarit permettant de saisir un nouvel enregistrement. La valeur 0 associée à *NumPersonne* est la valeur par défaut : souvenez-vous qu'Access affecte toujours la valeur par défaut 0 aux champs numériques. Or *NumPersonne* est *Numérique/entier long*.

Si cette application existait réellement (je ne l'ai faite que pour exploiter les relations), j'aurais défini tous les champs avec soin. J'aurais donc bien pris garde d'enlever cette valeur par défaut car elle n'a aucun sens.

J'avais donc raison : ce livre n'a jamais été emprunté !

2. Intégrité référentielle

2A. Définition

C'est un outil d'une importance cruciale. Il ne s'utilise qu'avec les relations *Un–à-plusieurs*. Vous pouvez l'appliquer à la création de la relation ou ultérieurement (il faudra alors double-cliquer sur le lien représentant la relation).

Derrière un nom barbare se cache timidement un concept très simple et surtout intuitif. Pourtant, il passe souvent mal. Concentrez-vous bien ! La définition est la suivante❶ :

L'intégrité référentielle permet d'assurer la cohérence de vos données en interdisant les situations d'inconsistance provoquées par des valeurs de clé étrangère ne correspondant à aucune valeur de la clé primaire.

2B. Un exemple pour comprendre

Envisageons une facture sans client associé (c-à-d. le numéro de client dans la table *Facture* ne correspond à aucun numéro de client dans la table *Client*). C'est une situation dramatique pour l'entreprise : qui va la régler ? Cette situation peut se produire pour plusieurs raisons :

- le client a été effacé de la table *Client* (l'enregistrement correspondant a été supprimé) ;
- l'identifiant du client a été modifié dans la table *Client* ;
- lors de l'enregistrement de la facture, un numéro de client erroné a été saisi.

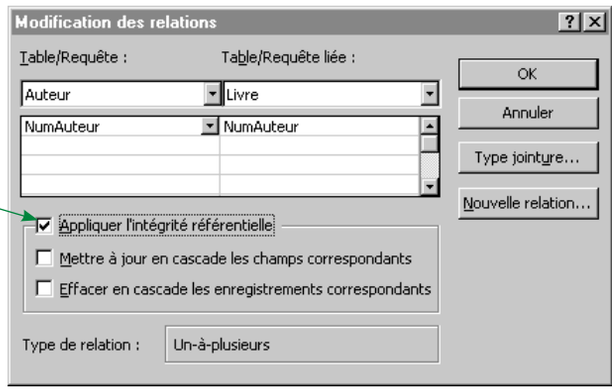
Avez-vous bien compris le principe ? Si je dis que le livre *American Psycho* est écrit par l'auteur 38 et que je n'ai pas d'auteur 38… rien ne va plus. Si le chien Démon a comme maître le propriétaire 11 et que ce propriétaire n'existe pas… Ces situations sont **forcément** fausses puisque les données ne sont pas cohérentes.

On retrouve ce que nous avons vu dans le paragraphe 4 de la séquence 3 : les différentes propriétés des champs permettent de limiter au maximum les erreurs de saisies. Elles ne sont donc pas obligatoires en théorie, mais le sont en pratique. De même, l'intégrité référentielle permet d'éviter l'inconsistance (facture sans client, chien sans maître…). Techniquement, si l'utilisateur ne fait jamais d'erreur, il ne se placera jamais en situation d'inconsistance. On n'est donc en théorie pas obligé d'utiliser cet outil. En pratique, bien entendu, il ne faut pas laisser à l'utilisateur la possibilité de se mettre en situation d'inconsistance (supprimer un client ayant des factures, entrer une facture avec un mauvais numéro de client…). On ne doit **jamais** faire confiance à l'utilisateur. Il faudra donc **toujours** appliquer l'intégrité référentielle.

❶ On va retrouver pour la première fois dans ce cours les notions à apprendre présentées en caractères verts. À l'examen, un candidat tombant sur la compétence base de données à l'épreuve de pratique des techniques informatiques sera forcément interrogé sur l'intégrité référentielle, sans laquelle la base n'est pas correcte.

2C. Et bien… appliquons l'intégrité référentielle

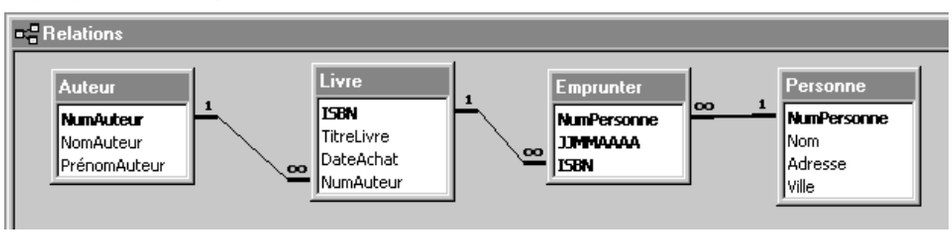
À la création d'une relation ou plus tard en double-cliquant sur son trait, on a accès à la case à cocher (que nous cocherons toujours) *Appliquer l'intégrité référentielle.*



Une fois la case cochée, je ne peux plus :

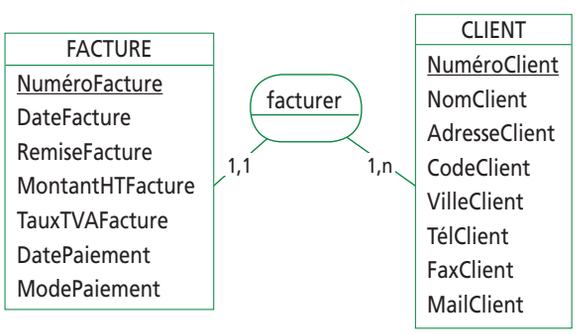
- ajouter un livre en lui associant un auteur non référencé dans ma base ;
- supprimer un auteur dont je possède des livres.

Si j'essaie de le faire, j'obtiendrai un message d'erreur.
Nous appliquons l'intégrité sur chaque relation. On obtient alors ce qui suit :

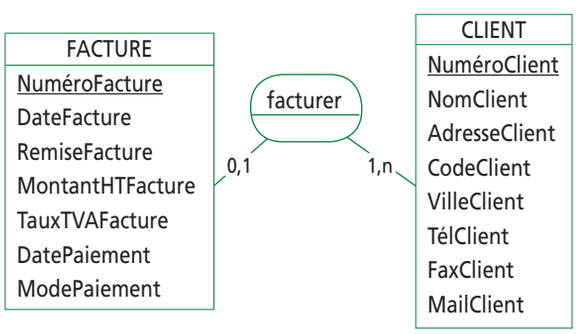


Les symboles *1* et ∞ correspondent aux *1* et *n* que vous connaissez depuis le MCD. Vous noterez que leur position est inversée vis-à-vis du MCD. C'est la notation américaine. Cela n'a aucune importance, la position des cardinalités est tout à fait arbitraire.
En fait, ce sont les cardinalités du MCD qui doivent vous indiquer s'il faut ou non appliquer l'intégrité référentielle. Observons les deux MCD ci-dessous.

Premier MCD



Second MCD



La différence tient dans la cardinalité de *facturer* sur *Facture*. Le premier MCD est correct. Il impose l'intégrité référentielle pour assurer qu'à une valeur de la clé étrangère *NuméroClient* dans *Facture* corresponde une valeur de clé primaire *NuméroClient* dans *Client*. En clair, l'intégrité impose qu'un client unique soit associé à toute facture. On retrouve bien la cardinalité *1,1*.

Le second MCD, erroné, permet qu'une facture ne soit associée à aucun client. Dans ce cas, nous n'avons pas d'intégrité. Mais, je le répète, ce MCD ne traduit pas la réalité.
L'intégrité référentielle permet donc d'assurer la correction de la base développée vis-à-vis du MCD. C'est un outil indispensable et pas un simple gadget !

Nous venons de voir les trois cas où l'inconsistance pouvait se produire :

- à la saisie des données (vous saisissez une nouvelle facture et votre numéro de client associé est faux) ;
- à la modification ou à la suppression des données (vous supprimez un client ou modifiez sa clé primaire alors qu'il possède des factures).

Comment régler ces trois problèmes ?

1. Saisie des factures avec un numéro de client erroné
La solution est pragmatique. On l'appliquera avec les formulaires car ils permettent une saisie graphique des informations. L'idée est de ne pas mettre de zone de texte où l'utilisateur saisit un numéro – il n'est d'ailleurs pas raisonnable d'espérer que l'utilisateur connaisse ces numéros par cœur –, mais d'utiliser une liste modifiable où l'utilisateur devra choisir parmi les clients référencés. Impossible donc de saisir une valeur qui n'est pas disponible.
Lancez Word et regardez la zone de liste vous permettant de saisir la police de caractères : c'est la même chose. Impossible de choisir une police inexistante !

2. Modifier un numéro de client
Il s'agit de modifier la valeur de la clé primaire, ce qui ne devrait pas arriver car une des caractéristiques de l'identifiant est sa stabilité. Cela peut néanmoins se produire dans la vraie vie si l'identifiant d'un produit est son nom : si ce dernier change, il faudra mettre à jour la clé primaire (cela milite d'ailleurs pour l'emploi de numéros identifiants dans tous les cas).
Changer la clé primaire impose de changer toutes les clés étrangères associées. Par exemple, imaginons le MLD suivant (il est partiel) modélisant la facturation d'une entreprise de VPC :

Produit (NomProduit, PrixProduit…)
Commande (NumCommande, NumClient…) primaire, étrangère#
Contenir (NumCommande#, NomProduit#, Quantité)

Si le produit *Buffet bas* est renommé *Buffet bas (acajou)*, tous les enregistrements de *Contenir* référençant le produit *Buffet bas* devront être modifiés pour référencer *Buffet bas (acajou)*. (Si l'identifiant de *Produit* avait été un numéro et non le nom, ce problème ne se serait pas posé.)

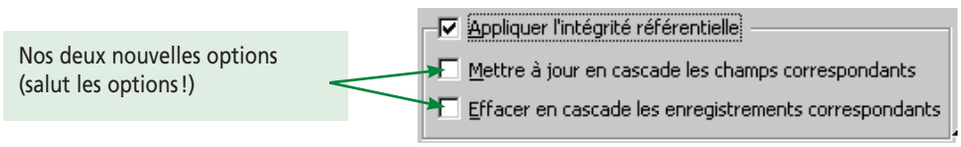
3. Supprimer un client
Il faut supprimer les factures associées au client.

Le premier problème est réglé par le choix d'un composant visuel adapté. Les deux autres requièrent au contraire un travail dans la base : mettre à jour les factures ou les supprimer.
Imaginez maintenant une base plus volumineuse avec beaucoup plus de liens clés primaires/clés étrangères. Par exemple, une facture possède plusieurs règlements, relances… Bref, supprimer un client oblige à supprimer ses factures, ce qui oblige à supprimer les règlements et relances correspondants. Bref, on souhaite supprimer un enregistrement dans une table et l'ensemble des tables s'en trouvent mises à jour par un effet boule de neige indispensable puisque, je vous le rappelle, l'objectif est d'éviter l'inconsistance.

Il est évident que dans ce cas où toutes les tables sont liées, il devient très délicat de faire la suppression à la main. C'est pourquoi l'activation de l'intégrité référentielle évite l'inconsistance de deux façons différentes, dépendant des options choisies :

- en empêchant toute modification ou suppression de données engendrant des incohérences. C'est ce que nous avons vu jusqu'à présent. Vous devez donc faire vous-même le travail. Il faudra par exemple supprimer les factures avant le client associé ;
- en propageant toutes les modifications que vous faites. La mise à jour boule de neige de la base est faite automatiquement.

Quand l'intégrité référentielle est activée, deux nouvelles options deviennent accessibles :



Elles permettent ce qui était interdit plus tôt (suppression ou modification de clé primaire) en mettant à jour les clés étrangères.

1. Mettre à jour en cascade les champs correspondants
Si vous changez la valeur d'un champ clé primaire, toutes les valeurs clés étrangères correspondantes sont également changées. Cela ne sert à rien quand la clé primaire est un *NuméroAuto* puisque ce numéro, géré par Access, n'est pas modifiable.
Exemple :
Si vous changez le nom d'un produit (ce nom étant clé primaire), il sera mis à jour dans toutes les factures.

2. Effacer en cascade les enregistrements correspondants (à manier avec précaution)
Dans la relation *1–plusieurs*, si vous supprimez le *1*, le *plusieurs* est automatiquement supprimé. On remarquera la dangerosité de cette option : l'utilisateur ne se rend pas forcément compte qu'en supprimant juste un client, il va supprimer en fait toutes les données associées (factures, règlements, relances…).
Exemple :
Si vous supprimez un client, toutes ses factures sont automatiquement supprimées. C'est très dangereux ! On préférera une solution obligeant l'utilisateur à supprimer d'abord toutes les factures puis le client. Cela se fera par programmation.
Si l'on tient à utiliser la suppression en cascade, il faudra impérativement afficher un message d'avertissement rappelant que la suppression de la facture entraînera celles des données associées (client, règlements…) et demandant une confirmation.

Si vous appliquez l'intégrité référentielle alors que vos tables contiennent déjà des données, Access vérifie leur cohérence. Si l'intégrité n'est pas respectée, vous aurez un message d'erreur et il faudra corriger les données avant d'appliquer l'intégrité. Si une telle mésaventure vous arrive, cela signifie que vos données n'ont aucun sens… c'est bien la preuve que l'intégrité référentielle est utile.
En général, on coche la modification en cascade mais pas la suppression.
En fait, on ne supprime généralement pas d'information d'une base. Supposons un chauffagiste proposant des contrats d'entretien de chaudières. Quand un contrat se termine et n'est pas renouvelé, on ne le supprimera pas de la base car il apporte des informations importantes. De plus, il est naturel de garder le client dans la base en tant qu'ancien client.

À la suppression, on préférera une propriété dans l'entité *Contrat* (donc un champ dans la table *Contrat*) indiquant s'il est terminé ou non (au choix un booléen ou une date de fin qui sera renseignée si le contrat est terminé). Le 20 décembre, on pourra ainsi envoyer un mailing à tous les clients qui n'ont pas renouvelé leur contrat arrivant à échéance au cours de l'année pour leur proposer une offre promotionnelle *spécial Noël* (avec le slogan *chaudière révisée, Noël tempéré !* ou *Offre de Noël, c'est un cadeau du ciel !* ou autre).

2D. Modification ou suppression d'une relation

Pour modifier une relation (notamment pour appliquer ou supprimer l'intégrité référentielle), double-cliquez sur le trait qui la matérialise. Vous arrivez alors dans la fenêtre permettant de la modifier.
Pour supprimer une relation, sélectionnez-la en cliquant sur son trait ; appuyez ensuite sur la touche *Suppr*. Après confirmation, votre relation est supprimée.

Exercice

Créez la relation entre les tables *Auteur* et *Livre* en prenant soin d'appliquer l'intégrité référentielle. Si vous avez rentré des données incorrectes au mépris de ma recommandation de données cohérentes, vous devrez les corriger avant de pouvoir l'appliquer. En effet, elle ne peut être activée que si les numéros d'auteur dans la table *Livre* existent dans la table *Auteur*.

2E. Attention, les enseignants peuvent se fâcher

Si vous voulez éviter de rendre un enseignant d'informatique en colère (lors d'un oral par exemple), ne montrez **jamais**, sous aucun prétexte, les relations lorsque l'on vous demande le MCD.
En particulier, ne mettez pas dans votre note de synthèse une copie écran de la fenêtre des relations à la place d'un MCD. Ce n'est pas la même chose ! D'ailleurs, je vous rappelle que si vous avez respecté ce cours, vous aurez toujours fait un MCD avant de commencer un travail sur Access. Montrer les relations fait croire à l'absence de MCD… aie aie !

Le MCD est un document de travail et de communication. Les relations, c'est un outil technique propre à Access permettant de créer facilement les relations et d'appliquer l'intégrité référentielle.

Synthèse

Cette séquence est très importante ; enfin, toutes le sont, mais celle-là particulièrement car l'étudiant (le débutant sous Access) s'en passe aisément.
En effet, si vous ne tenez pas compte de la deuxième séquence et ne créez pas de table… il est clair que vous ne pourrez rien faire. Difficile donc d'oublier cette partie du cours. Idem pour les requêtes, les formulaires ou les états.
En revanche, si l'on omet les relations et l'intégrité référentielle, on a l'impression (mais l'on se trompe) que tout va bien. Cela ne nous empêche pas de faire les formulaires et tout le *tra la la*. Cependant, les requêtes ne renverront pas toujours le bon résultat et l'application ne sera pas sécurisée au niveau de la cohérence des données.

Ne mélangez pas les relations et l'intégrité référentielle, ce sont des choses tout à fait distinctes :

- les relations permettent d'identifier les clés étrangères vis-à-vis des clés primaires ;
- l'intégrité référentielle, qui s'appuie sur une relation, assure la cohérence des données en interdisant que des valeurs de clé étrangère n'aient pas de contrepartie en clé primaire.

L'intégrité permet en outre d'automatiser la mise à jour des clés primaires vis-à-vis des clés étrangères :

- si l'on modifie une valeur de clé primaire, les clés étrangères sont mises à jour ; je vous rappelle néanmoins que cette situation ne doit pas arriver puisqu'un identifiant est stable. Je préfère donc que vous utilisiez constamment des numéros automatiques et n'activiez pas cette option dès lors sans objet ;
- si l'on supprime un enregistrement, la valeur de sa clé primaire disparaît avec. Par cohérence, tous les enregistrements ayant cette valeur en clé étrangère seront également supprimés. Vu le potentiel destructeur de cette option, il est indispensable de prévenir l'utilisateur de l'effet boule de neige et de lui demander confirmation avant la moindre suppression.

À l'issue de cette séquence, vous devez pouvoir :

- expliquer ce qu'est une relation, à quoi elle sert et comment la mettre en place ;
- idem pour l'intégrité référentielle ;
- idem pour les options de l'intégrité référentielle.

Si ce n'est pas le cas, relisez-la.

Remarque : un développeur passant l'épreuve de pratique informatique sur la compétence base de données ne coupera évidemment pas à des questions sur l'intégrité référentielle.

Séquence 5

Génération automatique de tables et import/export de données

Dans cette séquence, nous allons refaire le travail des séquences 2, 3 et 4 de façon automatisée : c'est un AGL (ici, PowerAMC) qui va générer la base à partir du MCD. Nous verrons également comment récupérer des données pour les mettre dans les tables Access.
Vous ne disposez sans doute pas d'un AGL (WinDesign, PowerAMC…) chez vous. L'important est de lire attentivement cette séquence pour comprendre le principe des manipulations.
Passer par un AGL pour générer une base n'est en fait intéressant que sur de très gros projets, ce qui n'est pas votre cas, ni le mien. De plus, il n'y a pas de miracle : en passant par un AGL, il faudra faire dans cet outil ce que vous faisiez sous Access.

▶ *Capacités attendues*

- **Savoir mener le cycle de développement MCD → MLD → Tables Access avec PowerAMC**
- **Importer des données dans Access**
- **Exporter des données**

▶ *Contenu*

1. Rôle de l'AGL

Un AGL (Atelier de Génie Logiciel) est un outil (logiciel) assistant l'analyste et le développeur. Il permet notamment :

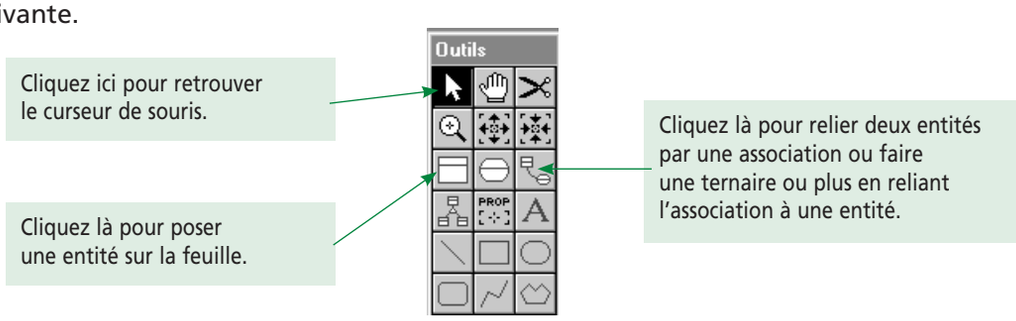
- de réaliser l'analyse (MCD) ;
- de faire le lien analyse/développement (MLD, génération de la base de données) ;
- de générer plus ou moins efficacement du code (réalisation de formulaires manipulant les tables).

Il y a autant de définitions et de fonctionnalités que d'AGL. La génération automatique du code est un outil très intéressant. En pratique, il ne faut pas trop en attendre : le développeur n'est pas remplacé ! L'AGL que nous allons utiliser ici est PowerAMC de la société Sybase.

2. Utilisation de l'AGL

2A. Création du MCD

En lançant PowerAMC données, vous obtenez une feuille vierge avec la boîte à outils suivante.



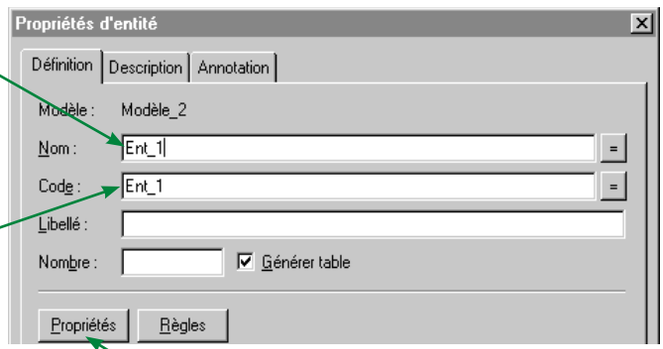
Le principe est simple : vous placez les entités et les associations et vous définissez les propriétés.
Je vais faire le MCD représentant les livres et les auteurs (soit deux tables reliées par une association).
Dans un premier temps, je place entités et association.

Je double-clique ensuite sur les différents composants (les entités et l'association) pour définir les noms et les propriétés.

En double-cliquant sur l'entité *Ent_1*, j'obtiens la fenêtre suivante :

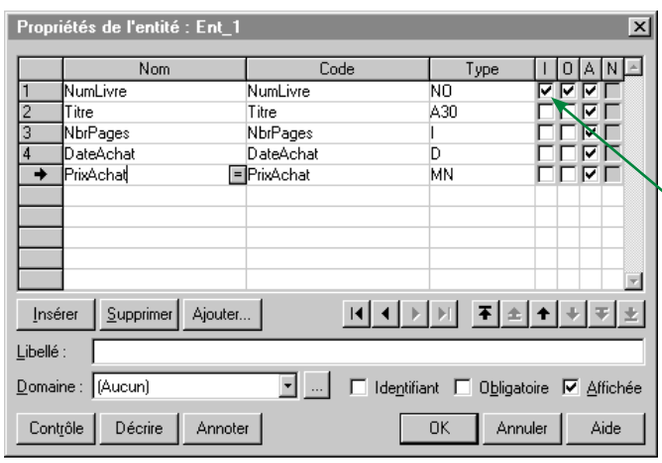
Nom de l'entité (mettez *LIVRE*)

Nom interne de l'entité.
C'est lui qui sera utilisé lors de la génération du MLD.
Tapez *LIVRE* ou, mieux encore, cliquez sur le = en fin de ligne.
Le nom entré dans la ligne précédente sera automatiquement repris.



Cliquez ici pour définir les propriétés de l'entité.

Après avoir saisi le nom et le code, cliquez sur *Propriétés*. Vous obtenez la fenêtre suivante, permettant de définir complètement les propriétés (nom, type et identifiant).



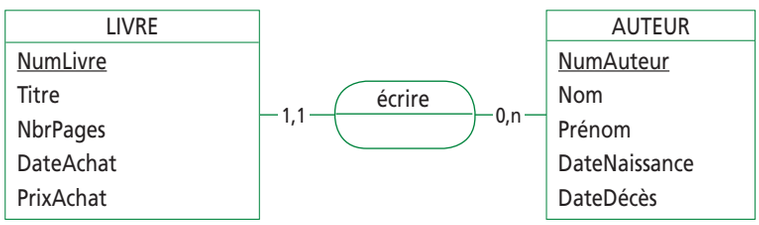
I signifie identifiant. *NumLivre* sera donc l'identifiant (souligné) de l'entité.

Si, en cliquant sur le « = » du code, vous obtenez le nom de la propriété en majuscules, réglez votre logiciel : *Fichier/Options du modèle* onglet *Code*, cochez *Majuscule et minuscule*.
La colonne *Type* indique évidemment le type de la donnée. Lorsque vous êtes dans la zone de type, cliquez sur le bouton … et choisissez parmi les nombreuses options. Vous remarquerez que dans la fenêtre ci-dessus, j'ai défini :

- *NumLivre* de type séquentiel ;
- *Titre* comme du texte de longueur 30 ; parmi les différents types texte, j'ai utilisé *Caractère alpha*. En effet, les autres génèrent sous Access un type *Mémo* et non texte.
- *NbrPages* comme un entier.

Pourquoi les types proposés ne correspondent pas exactement aux types d'Access ? Tout simplement car un AGL n'est pas dévolu à un outil de développement particulier ! À partir de PowerAMC, vous pouvez générer des bases pour de multiples SGBDR.
Pour une description des types, n'hésitez pas à cliquer sur *Aide* dans la fenêtre correspondante !

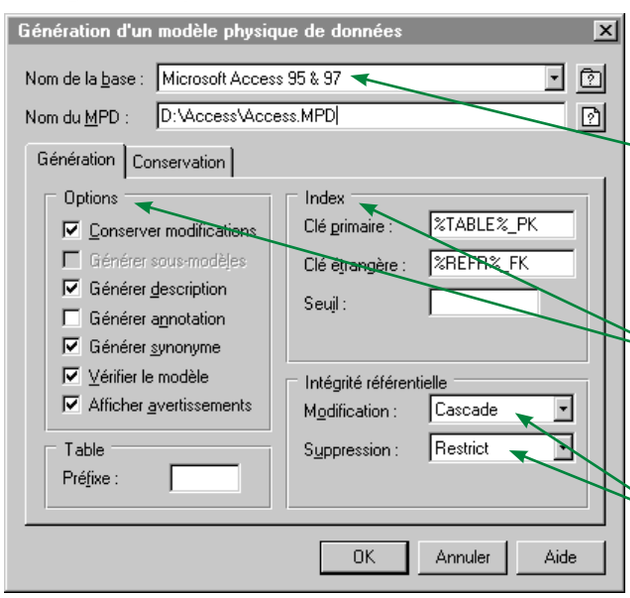
En double-cliquant sur l'ovale de l'association, vous obtiendrez la même fenêtre (à ceci près que ses propriétés ne peuvent être définies en tant qu'identifiant).
En double-cliquant sur les traits (pattes) des associations, vous pouvez modifier les cardinalités, définir une entité faible❶ ou donner un rôle à la patte❷.
Bien. Définissez les différentes propriétés pour obtenir le MCD final :



DateAchat, *DateNaissance*, *DateDécès* sont de type *date*, *PrixAchat* de type *monnaie*, *Nom* et *Prénom* sont du texte de longueur 20.
Pour donner un nom parlant à notre modèle, menu *Dictionnaire/Propriétés du modèle*, zones *Nom* et *Code*, mettez *Biblio*.

2B. Génération du MLD

Je vous ai dit dans le cours d'analyse que la génération du MLD était tellement simple (dans la mesure où il suffit d'appliquer des règles très précises) que les AGL pouvaient le générer automatiquement. Eh bien…
Lancez le menu *Dictionnaire/Générer le modèle physique*. Vous obtenez une boîte de dialogue. Remplissez-la ainsi :



Nom de la base, choisissez Microsoft Access 2003. Vous remarquerez le nombre de SGBD proposés ! Notez que *Nom de la base* n'est pas très heureux car ce n'est pas le nom de la base de données (*bibliothèque.mdb* par exemple) mais le nom du SGBD.

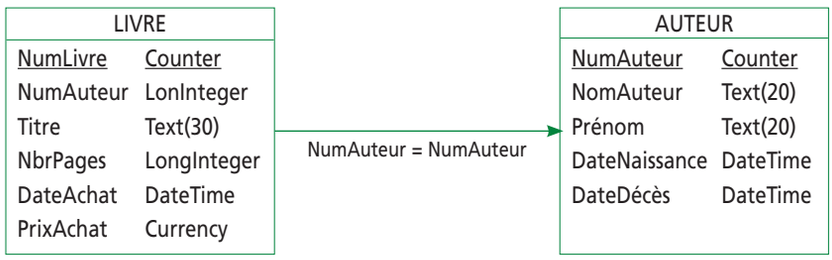
Laissez les valeurs par défaut.

Comme nous l'avons dit dans la séquence précédente, le plus raisonnable est d'appliquer en cascade les modifications mais d'interdire les suppressions.

❶ *C'est ce qu'AMC appelle* Lien identifiant. *Sous AMC, on utilise la symbolique de la mise entre parenthèses de la cardinalité 1,1 (revoir le cours d'analyse, séquence 3, paragraphe 3C).*

❷ *Utile lors d'associations réflexives (même cours, même séquence, paragraphe 2).*

En cliquant sur *OK*, vous lancez la génération du MLD, qui donne :

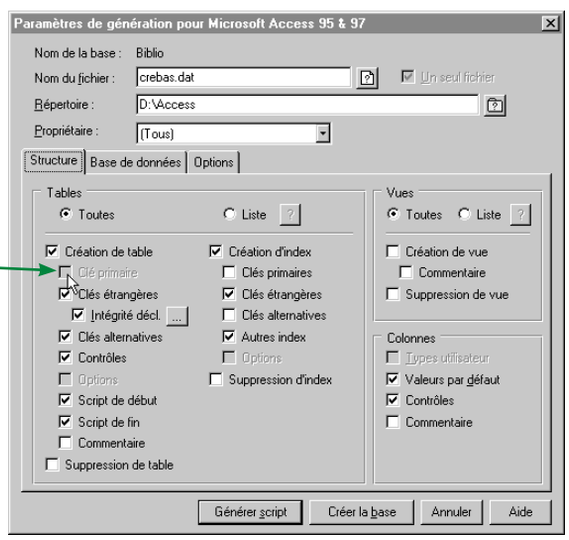


Ce formalisme est nouveau pour vous… c'est la même chose que ce que nous avons étudié ensemble dans le cours d'analyse. Seule la présentation, graphique, change. Double-cliquez sur la flèche du milieu. Vous verrez qu'elle représente bien le lien clé primaire/clé étrangère.
Vous obtenez également une fenêtre *Message* qui détaille les opérations faites. Vous remarquerez que le MCD a été vérifié… houa, super, vous avez un prof à la maison ! En fait, ne vous illusionnez pas : la vérification est purement syntaxique, pas sémantique.

2C. Génération de la base depuis Access

Menu *SGBD/Générer la base de données*… Le principe est simple : AMC va créer un script (une suite de commandes) SQL compatible avec Access et créant les différentes tables et relations. Vous obtenez la fenêtre de paramétrage suivante :

Nous avons un gros problème ! Il est visiblement impossible de définir les clés primaires dans les tables :
la case est décochée et grisée.



Que se passe-t-il ? Ma foi, je n'en sais rien. La version de PowerAMC que j'utilise (6.1.0) ne colle visiblement pas parfaitement avec Access. La documentation nous dit d'ailleurs : « la disponibilité de ces paramètres dépend de la base de données cible. Les paramètres non disponibles s'affichent en grisé et ne peuvent pas être sélectionnés. ».
Je suis coincé. Si vous utilisez une version plus récente de PowerAMC, possédant notamment un pilote pour Access 2003, vous pourrez sans doute générer les clés primaires.
Je ne vais pas plus loin dans cette génération. Vous noterez quand même les deux boutons permettant de générer la base :

- *Générer script* va créer un fichier contenant du code VBA. Vous devez alors créer la base Access, créer un module, y importer ce fichier et l'exécuter. Les tables et les relations doivent alors être créées ;

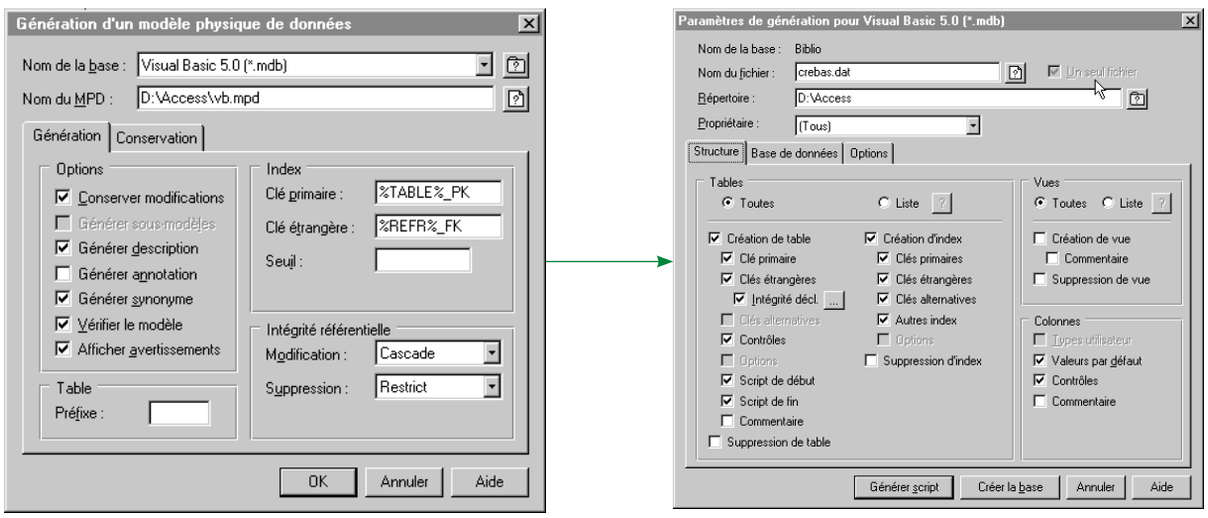
- *Créer la base* vous permet de vous connecter à la base Access (qui doit exister). PowerAMC exécute les instructions SQL créant les tables via la connexion.

Comment allons-nous faire pour générer notre base ? Nous allons conserver la même méthode, mais en passant par un script VB. Et là, cela fonctionne !

2D. Génération de la base en passant par VB

On reprend ce que l'on a fait en 2B. Partant du MCD, vous faites *Dictionnaire/Générer le modèle physique…* Dans *Nom de la base*, on ne choisit plus *Access* mais *Visual Basic 5.0*❶. Le reste des options ne change pas.

En validant, on arrive de nouveau dans la fenêtre de paramétrage… cette fois, vous remarquez que la génération des clés primaires est proposée.



Nous pouvons maintenant créer la base. Pour cela, nous cliquons sur *Générer script*. On obtient alors deux choses :

- la fenêtre *Message* qui indique que tout s'est bien passé et fournit la marche à suivre pour générer la base à partir du script ;
- PowerAMC vous propose de visualiser le script. Faites-le ; vous verrez la définition des tables, des index et de la relation.

Reprenons les étapes de la fenêtre *Message* et faisons-les :

(1) Lancer Microsoft Visual Basic Entreprise 5.0.

Aucun problème !

(2) Créer une nouvelle base de données Access à l'aide de Visual Data Manager (menu Add-Ins).

En fait, c'est *Compléments/Gestionnaire de données…* dans la version française de VB. Dans la barre de menus de cet outil, *Fichier/Nouvelle base de données…/Microsoft Access/MDB version 7.0* (cela correspond à Access 97 ; si votre VB vous propose mieux, prenez !).

❶ *Ma version de PowerAMC est de la même époque que VB 5 et Access 97. Si vous avez une version plus récente, vous aurez accès à VB 6 voire VB.Net et Access 2003. De toute façon, la compatibilité ascendante fait qu'une base Access 97 (ou 95 ou 2) peut toujours être convertie en une base plus récente. Donc RAS !*

Appelez votre base *Biblio.mdb* (c'est le nom du modèle défini à la fin du paragraphe 2A). Il est important de donner ce nom de modèle car PowerAMC l'a mis en dur dans le script. Si la base s'appelle autrement, elle ne pourra être générée.

Fermez ensuite impérativement l'outil de gestion des données, sinon le script VB ne pourra pas accéder à la base.

(3) Ouvrir le projet d:\access\crebas.vbp.

Sous VB, menu *Fichier/Ouvrir un projet…* et sélectionnez ce projet (dans le dossier et sous le nom que vous avez rentré dans la fenêtre de paramétrage sous PowerAMC !).

(4) Exécuter le programme (menu Run, option Start).

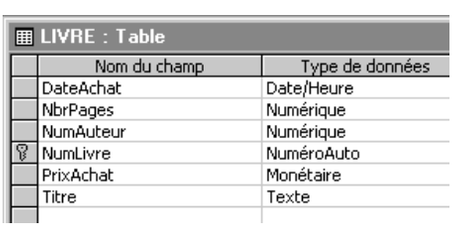
À l'exécution, l'application ouvre une fenêtre *Ouverture de la base de données*. Sélectionnez la base créée dans l'étape (2). Validez… la fenêtre *Exécution* vous indique ce qui est fait :

- ouverture de la base ;
- création de la table *Auteur* puis de sa clé primaire ;
- création de la table *Livre* puis de sa clé primaire ;
- création de la relation entre les deux tables.

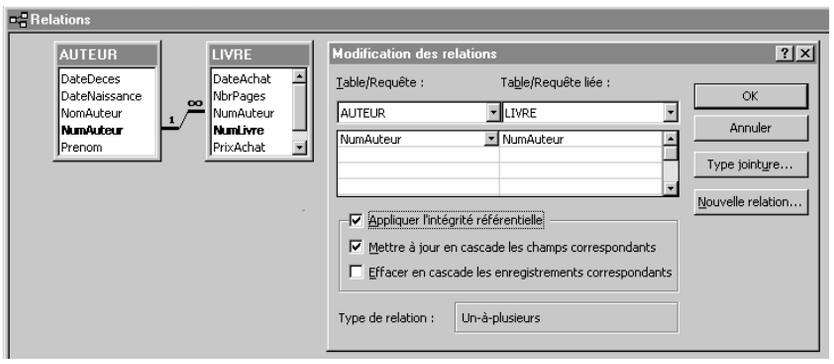
Nous pouvons maintenant visualiser la table : soit vous passez par le *Gestionnaire de données* et ouvrez la base, soit vous utilisez Access.

Je vais utiliser Access. Je quitte donc VB sans enregistrer puisque le script ne me sert plus à rien. Je double-clique sur *biblio.mdb* et j'obtiens un message m'indiquant que la base provient d'une version antérieure d'Access. C'est vrai puisque la version 7.0 sélectionnée dans l'étape (2) correspond à Access 97. Si vous avez une version plus récente du gestionnaire de données, vous avez sans doute généré directement une base Access 2003.

Dans notre cas, il suffit de choisir *Convertir une base de données*. J'appelle la nouvelle base *Biblio 2003.mdb*. La base est conforme à mes attentes : tables et index ont été correctement définis. La relation entre *Livre* et *Auteur* est correcte. On a bien une modification (et pas suppression) en cascade. Je vous présente ci-dessous quelques copies d'écran du résultat :



Enfin, vérifions la relation :



2E. Allons plus loin

Observez que le champ *NomAuteur* de *Auteur* est bien du texte de longueur 20 ! (Idem pour les autres champs texte.)

PowerAMC permet de définir finement les champs dans le MLD (conditions de validité, valeur obligatoire ou non…). Cela se retrouvera sous Access dans l'onglet *Général* pour la description fine des propriétés des différents champs.

PowerAMC propose quelque chose d'intéressant pour tester le comportement d'une base (ou pour écrire des requêtes). Il s'agit de la génération de données de test. L'intérêt est que le logiciel va générer des données aléatoires mais cohérentes, c'est-à-dire respectant les relations donc le lien clé primaire/clé étrangère. Cela dépassant le cadre de notre étude, je vous donne simplement les différentes étapes à suivre :

1. Partez du MLD.
2. Menu *Dictionnaire/Profils/Liste des profils…* Cela va vous permettre de définir les types de données à générer (et une plage de valeur) pour chaque champ.

   Créez un profil *Date* de classe *Date/Heure*, *Entier* et *Texte* de classe *Nombre* et *Caractère*. Pour chaque profil, cliquez sur *Propriétés* pour définir les plages des données puis sur *affecter* pour attribuer le profil aux champs concernés.
3. Vous pouvez lancer la génération : *SGBD/Générer les données de test…*

3. Importation et exportation des données

3A. Introduction

Nous venons de voir comment automatiser la création d'une base à partir d'un AGL. Nous allons maintenant voir comment récupérer, toujours automatiquement, des données externes pour les importer dans Access.

L'idée est la suivante : très souvent, lorsque l'on crée une base de données, certaines données existent déjà dans le système d'information de l'entreprise :

- soit l'application que vous écrivez existait déjà de façon plus ou moins efficace (faite sous Excel, avec des fichiers…) ;
- soit vous pouvez récupérer les données clientèle, comptabilité ou autres depuis des outils tiers (logiciels de gestion commerciale, de comptabilité…).

Ces logiciels proposent généralement d'exporter leurs données, au moins sous forme texte. Access, lui, vous permettra d'importer ces données dans la base.

Ceci est très important. Vous ne pouvez le mesurer qu'en l'ayant déjà fait : il n'y a rien de plus long et de plus rébarbatif que de saisir des données. Saisir un ou deux enregistrements, c'est amusant. En rentrer plusieurs centaines pour intégrer tous les clients de l'entreprise, c'est un cauchemar. Le seul cas où vous devrez le faire à la main sera l'absence totale d'informatisation préalable.

3B. Mécanisme

C'est très simple car très automatisé. Menu *Fichier/Données externes/Importer…* Vous choisissez ensuite le fichier contenant les données que vous voulez récupérer. Par défaut, la fenêtre vous propose des bases de données Access mais vous pouvez changer dans *Type de fichiers* pour récupérer tout type de données (classeurs Excel, fichiers texte, autres bases de données…).

Si vous importez les données depuis une autre base, vous pourrez importer tout type d'objet, y compris des formulaires, des états ou des modules. Si vous importez vos données depuis un classeur Excel, vous pourrez choisir les colonnes à importer en spécifiant un nom et un type de champ pour chacune. De même, si les données viennent d'un fichier texte, vous pourrez spécifier le type de chaque champ et le séparateur utilisé (tabulation, point-virgule…).

Faites un essai d'import. Vous verrez que la démarche est intuitive. Le devoir en autocorrection de ce cours contient un import à réaliser.

3C. Export de données

Par symétrie, il est normal qu'Access propose d'exporter des données (le contenu d'une table ou le résultat d'une requête). Cela permet de récupérer des données en vue de leur traitement dans une autre application.

Par exemple, imaginez une base Access stockant les cours de bourse des actions. Les cours seront exportés d'Access puis importés sous Excel pour réaliser des graphiques et des statistiques. Mieux encore, on peut piloter l'application Excel depuis Access. Cela donne un logiciel très intégré : l'utilisateur choisit son action et une plage de dates, appuie sur un bouton et hop ! La courbe s'affiche. L'import et l'export sont réalisés de façon transparente par la magie de la programmation.

La manipulation est simple : lorsque l'objet à exporter est sélectionné, faites *Fichier/Exporter…* et laissez-vous guider.

Synthèse

Que retenir de cette séquence ? Si vous avez un AGL à votre disposition, c'est au mieux en version de démonstration. Il y a donc peu de chances que vous fassiez souvent la manipulation que nous venons d'étudier.

Finalement, le plus important est de retenir le principe et savoir que l'on peut exploiter l'AGL pour réaliser une partie significative du travail sous la base de données.

À l'issue de cette séquence, vous devez pouvoir :

- expliquer le rôle d'un AGL dans un cycle de développement ;
- savoir importer des données d'origines variées dans Access ;
- savoir exporter des données d'Access.

Si ce n'est pas le cas, relisez-la.

Séquence 6

Les requêtes

Dans cette séquence, nous allons voir comment rédiger des requêtes sous Access. Cette séquence est brève puisque cela a déjà été abordé dans le cours de SQL. Nous mentionnerons également le mode QBE.

▶ *Capacités attendues*

- **Exploiter pleinement le cours de SQL sous Access**

▶ *Contenu*

1. Introduction

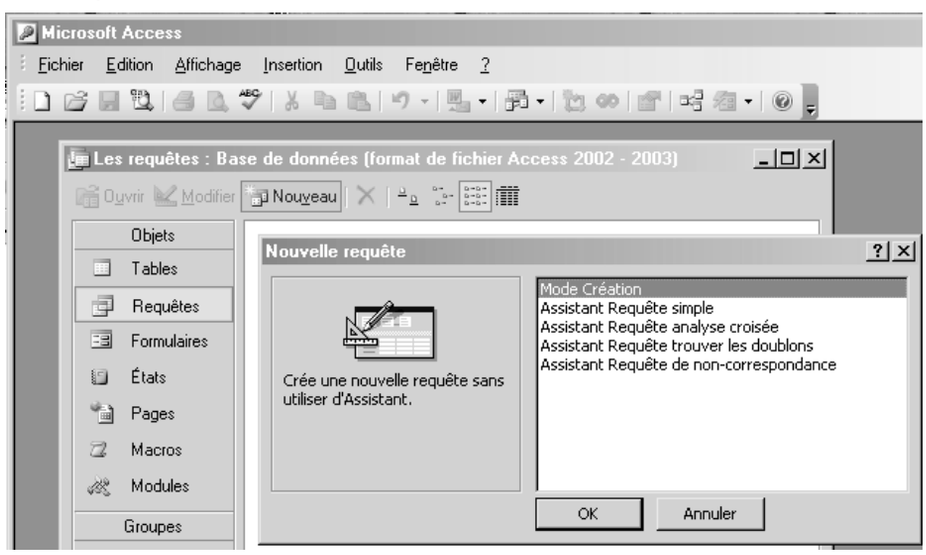
1A. Présentation

Pour aborder cette partie, vous devez avoir travaillé le cours de SQL. Je vous situe à nouveau le contexte : dans les séquences précédentes, nous avons appris à créer les tables et à les remplir.
Avec les requêtes, nous nous dotons d'un outil permettant de récupérer les données répondant à des critères précis. Nous savons donc trouver la bonne information parmi un ensemble volumineux.
Je ne referai bien entendu pas le cours de SQL ici. Je vais juste vous rappeler comment taper les requêtes SQL sous Access. Je vous présenterai également le mode QBE.

1B. Rappel : comment saisir une requête

1B1. Les objets *Requêtes*

Access offre la possibilité d'établir des requêtes de consultation ou de mise à jour des données. Pour créer une requête, on part de la fenêtre *Base de données* (*F11*) et on sélectionne les objets *Requêtes*, bouton *Nouveau* puis *Mode Création*. (Des assistants sont proposés pour écrire certaines requêtes classiques… Je vous conseille de les essayer à vos moments perdus.)



1B2. Le mode *Création*

Le problème, c'est que vous arrivez au *mode Création*, également appelé *QBE* (*Query By Exemple* pour *requête par l'exemple*). Ce mode permet d'écrire plus ou moins graphiquement une requête. C'est sensé être plus facile que l'écriture sous SQL. Mais :

- ce n'est pas si intuitif que cela ;
- toutes les requêtes ne peuvent pas être écrites sous la forme QBE ;
- dès que vous voudrez rédiger une sous-requête, il faudra l'écrire sous la forme SQL à l'intérieur de la requête principale en QBE ;
- pour un informaticien, il est invraisemblable de simplement oser avoir l'idée de ne pas utiliser SQL ou de témoigner un quelconque intérêt à l'existence même de

QBE.

En résumé, le mode *Création* est à proscrire. Comme on arrive hélas directement dedans, il faut se dépêcher d'en sortir pour arriver au mode *SQL*.

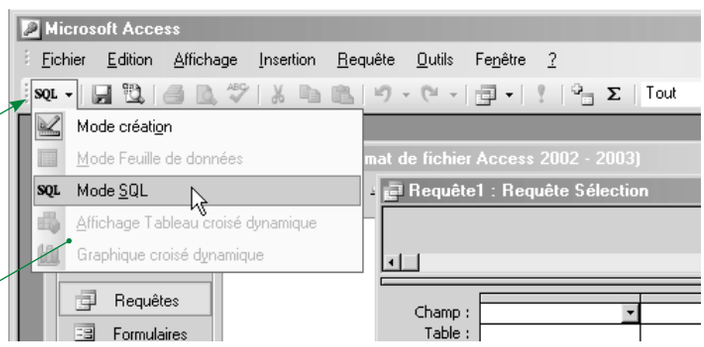
1B3. Le mode SQL et l'exécution de la requête

Une fois validé *Mode Création* dans la boîte de dialogue *Nouvelle requête*, vous obtenez une nouvelle boîte vous permettant de choisir les sources de données (tables ou requêtes) à utiliser pour votre requête. Comme notre objectif est de quitter ce mode *Création*, on ne choisit rien et on clique sur *Fermer*.

On peut du coup enfin accéder au menu *Affichage/Mode SQL* ou passer par le bouton de la barre d'outils *Création de requête*. On retrouve un bouton *Affichage* basculant comme l'illustre la copie écran ci-dessous.

Comme nous n'avons ajouté aucune table dans le mode Création, *le bouton* Affichage *propose par défaut le mode SQL. D'où son* ***icone.***

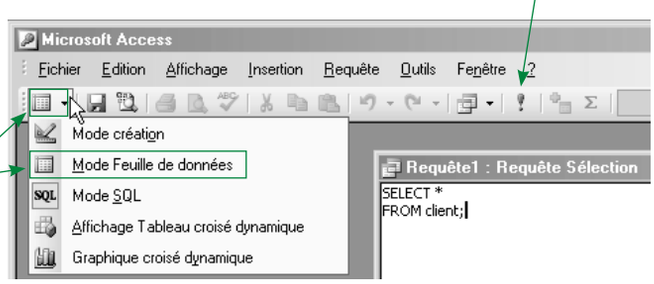
Si l'on ouvre les commandes du bouton, on voit que seuls le mode actuel (Création) *et le mode* SQL *sont disponibles.*



Vous basculez alors dans le mode coloré du SQL où votre potentiel pourra enfin s'exprimer. Pour ma part, j'ai tapé une modeste requête. Vous voyez ci-dessous que le bouton *Affichage* s'est actualisé et propose maintenant le mode *Feuille de données* qui permet d'afficher le résultat de la requête. Pour exécuter la requête, vous pouvez également faire *Requête/Exécuter* ou cliquer sur le bouton associé.

Bouton Exécution

Mode
Feuille de données



Voici la différence entre les commandes *Affichage/Mode Feuille de données* et *Requête/Exécution* :

- dans le cas d'une requête de sélection, les deux commandes sont équivalentes;
- pour les autres requêtes (ajout, modification et suppression), le mode *Feuille de données* affiche les enregistrements ajoutés, modifiés ou supprimés sans faire de traitement sur les tables. Cela permet de tester à blanc votre requête. La commande *Exécution* exécute la requête donc modifie réellement les données.

2. Élaboration des requêtes

Les requêtes peuvent être définies en SQL (*Structured Query Language*) ou QBE (*Query By Example*). Les deux sont sémantiquement équivalents (Access passe indifféremment d'un mode à l'autre). SQL sera néanmoins préféré car il est plus formel et plus puissant. De plus, on le retrouve dans tous les SGBDR. On laissera QBE aux non-informaticiens.

Ne pas passer d'un mode à l'autre ; en effet, si vous saisissez une requête SQL puis basculez par accident en mode *Création* avant de revenir en SQL, il y a un grand risque que votre requête ait été réécrite, et pas à votre avantage.

Démonstration ? Ma foi, vous en ferez sans doute l'expérience bien assez tôt…

2A. QBE

QBE est une interface permettant à l'utilisateur de définir des requêtes en sélectionnant les champs dans une grille d'interrogation à l'aide de la souris sans avoir à rédiger de commande SQL.
Pour accéder à l'interface QBE, objets *Requêtes*, bouton *Nouveau* puis *Mode Création.* Ajoutez les tables dont vous aurez besoin puis fermez la fenêtre *Ajouter une table*. Vous obtenez alors la fenêtre QBE.

Exemple
On veut la liste des factures (réduites aux numéros de facture et de client) telles que :

- le numéro du client est 3 et le total HT de la facture est supérieur à 1 000 €

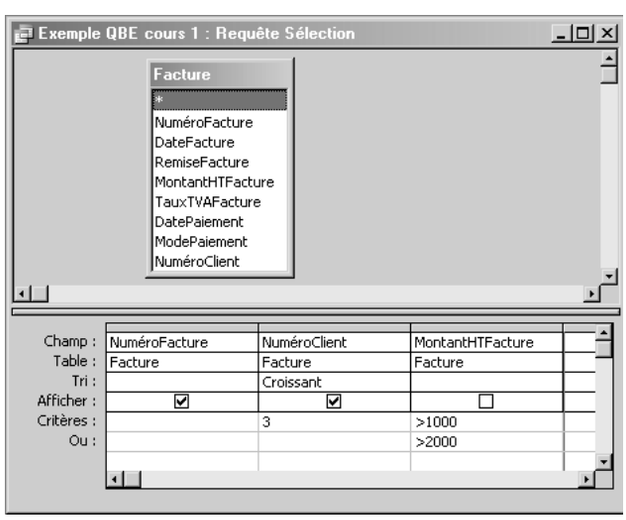
OU

- le total de la facture est supérieur à 2 000 €;

Le résultat doit être trié par numéro de client croissant.
La traduction SQL s'écrit :

```
SELECT NuméroFacture, NuméroClient
FROM Facture
WHERE (NuméroClient = 3 AND MontantHTFacture > 1000) OR (MontantHTFacture > 2000)
ORDER BY 2 ;
```

La traduction en QBE donnera :



Remarques

- observez l'utilité de la croix dans la ligne *Afficher* ;
- les critères s'écrivent de la même façon étrange que les règles de validité (voir séquence 3, paragraphe 3E) ;
- les critères *Ou* se lisent ligne à ligne (*ligne1 Ou ligne2*…), les critères *Et* sur la même ligne ;
- QBE et SQL contiennent exactement les mêmes informations, seule la syntaxe diffère.

Voici quelques exemples de codification des critères :

Exemples de critères QBE	
On veut les enregistrements vérifiant	Écriture du critère en QBE
Montant supérieur à 100 euros	>100
Champ ayant la valeur *Bipsie*	="Bipsie"
Date antérieure au 5 mars 2009	<#05/03/2009#
Tous les noms à partir de « ferdinand » (ordre alphabétique, pas de distinction majuscules/minuscules)	>="ferdinand"
Champ non nul (contenant une valeur)	est pas null
Champ contenant la date du jour	=date()

Remarque

On retrouve les dates entre dièses (#) et les chaînes entre guillemets du SQL.

2B. SQL

Résumons ce que nous avons déjà vu.

Objets *Requêtes*, bouton *Nouveau* puis *Mode Création.* Cliquez *Fermer* sans ajouter de table… Le bouton d'affichage propose alors le mode SQL. (Je ne connais pas de façon d'accéder directement au mode SQL, sauf en supprimant le bouton QBE.) Access ouvre alors une fenêtre vierge où vous pouvez saisir votre requête.

Access gère indifféremment les modes QBE ou SQL pour une requête ; vous pouvez donc passer de l'un à l'autre. Toutefois, lorsque vous écrivez votre requête en QBE et demandez la traduction SQL, Access génère une requête SQL assez embrouillée. Évitez donc de faire SQL → QBE → SQL car vous ne retrouverez plus votre joli code !

2C. Exécution d'une requête

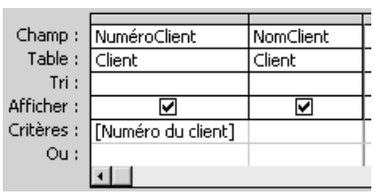
Il existe quatre méthodes pour exécuter une requête de sélection :

- après élaboration de votre requête, lancez la commande *Requête/Exécuter* ;
- cliquez sur le bouton de la barre d'outils (un point d'exclamation rouge) ;
- choisissez *Feuille de données* depuis le bouton d'affichage ;
- à partir de la fenêtre principale, sélectionnez les objets *Requêtes*, cliquez sur la requête qui vous intéresse puis *Ouvrir*.

2D. Les requêtes paramétrées

Une requête paramétrée est une requête dans laquelle au moins un critère dépendra d'une valeur saisie par l'utilisateur (et non de constantes ou d'autres champs de la base).
Par exemple, pour afficher le nom du client dont l'utilisateur fournira le numéro, vous devez réaliser au choix :

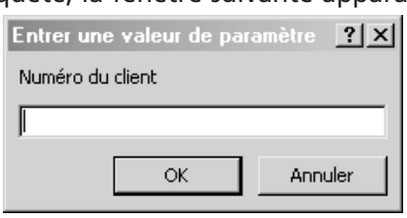
La grille QBE ci-dessous :



La requête SQL suivante :

```
SELECT NomClient
FROM Client
WHERE NuméroClient = [Numéro du client] ;
```

Lors de l'exécution de la requête, la fenêtre suivante apparaîtra :



Vous devrez saisir le numéro voulu, puis valider.

Remarque importante :
Vous avez parfois du mal à comprendre le fonctionnement des requêtes paramétrées. Je vous rappelle donc les deux points suivants :

1. certaines requêtes doivent être paramétrées (par exemple la recherche d'un client), d'autres non (par exemple : la liste des factures impayées) ;
2. comment Access sait-il qu'une requête est paramétrée et comment identifie-t-il les paramètres ? Autant les langages de programmation (VB, Delphi…) emploient un protocole assez compliqué pour gérer les paramètres, autant Access possède une technique d'une simplicité folle : dans la requête (clauses *select*, *where*…), tout ce qui n'est pas reconnu comme un champ d'une des tables de la clause *from* est réputé être un paramètre. Ainsi, si Access vous demande d'entrer la valeur d'un paramètre alors que votre requête n'est pas sensée être paramétrée, c'est juste que vous avez mal orthographié un nom de champ ou que la table le contenant n'est pas dans le *from*.

3. Les requêtes à la base des formulaires et états

Les formulaires permettent d'afficher (donc à destination de l'écran) les informations contenues dans la base. Les états ont la même fonction, mais sont à destination de l'imprimante. Dans les deux cas, ces informations peuvent être « brutes » (c-à-d. issues directement des tables) ou « raffinées » (provenant d'une requête, donc issues d'un traitement préalable des données brutes).

Chaque état ou formulaire possédera donc une source de données (soit table soit requête) dans laquelle il puisera ses données. Les données qui ne sont pas dans la source lui seront inaccessibles (sauf programmation, au programme de la seconde année).

Si un formulaire est basé sur une table, vous pourrez vous en servir pour rentrer des données dans cette table. Le seul intérêt de passer par le formulaire (et non par la table ouverte en mode *Feuille de données*) réside dans l'aspect visuel plus agréable : les fonctionnalités sont exactement les mêmes, au point que l'on peut ouvrir le formulaire en mode *Feuille de données* (mode texte) et non en mode graphique❶.

En pratique, les formulaires et états seront donc basés sur des tables ou des requêtes. L'utilisateur de la base (qui ne sera pas le concepteur et ne sera pas supposé avoir de connaissance particulière en informatique) ne connaîtra de la base que les formulaires et les états ; tables et requêtes lui seront cachées.

❶ *L'année prochaine, vous étudierez la programmation sous Access. Les formulaires vous permettront alors de réaliser des tâches complexes par le biais de boutons ou autre. Les formulaires sont donc en fait bien plus qu'un simple outil d'affichage ! Vous en aurez un petit aperçu dès cette année avec la séquence 13.*

Synthèse

Euh… que dire ?
Cette séquence se veut opérationnelle ; pour un cours sur SQL, il faut se plonger dans… le fascicule de cours du même nom !

SQL est ici un outil indispensable mais mineur. Indispensable car son objet est de fournir aux formulaires et aux états les sources de données qui les alimenteront mais mineur puisqu'on suppose SQL assimilé avant l'étude du SGBD.

Finalement, Access (et tout SGBD du même type) est un outil exigeant car il vous oblige à connaître :

- l'analyse pour réaliser des tables correctes ;
- SQL pour récupérer les données ;
- la programmation pour automatiser la base (cours de première année, séquence 12 de ce cours), voire réaliser des traitements (cours de seconde année) ;
- la programmation événementielle pour travailler sur les formulaires et les contrôles (c'est l'objet de la séquence suivante) ;
- bien entendu, les notions propres aux SGBD.

Séquence 7

Présentation des formulaires

Nous abordons maintenant le formulaire. Dans cette séquence, il s'agit d'étudier son principe et de voir tout ce qui gravite autour.

▶ *Capacités attendues*

- **Savoir créer des formulaires simples en exploitant des assistants**

▶ *Contenu*

1. Introduction

1A. Définition

Un formulaire est une fenêtre permettant de manipuler des enregistrements (consultation, saisie, modification et suppression). Mais bon, vous savez déjà faire cela avec les tables. Que va apporter le formulaire ? En fait, beaucoup de choses !
Un formulaire est graphique. Il offre une ergonomie sans commune mesure avec les tables. Vous pouvez utiliser des contrôles visuels (boutons, cases à cocher…) en fonction des données à saisir.
Un formulaire est généralement basé sur une source de données (table ou requête, voir le paragraphe 2B) dont il affiche les enregistrements. Il permet donc de visualiser et manipuler des informations complexes issues de plusieurs tables. Vous pouvez par exemple accéder à :

- un auteur et tous ses livres ;
- un client et toutes ses factures, avec pour chacune les règlements associés.

On peut finalement voir un formulaire comme l'exploitation graphique du résultat d'une requête.
Ce n'est pas tout ! On peut enrichir les formulaires de code VB❶. Par exemple, lorsque vous cliquerez sur le bouton *Suppression*, une procédure s'exécutera pour supprimer l'enregistrement courant. Arrivé à ce niveau, la difficulté est algorithmique. La partie programmation sera légèrement abordée dans ce cours (dans la douzième séquence). Vous irez plus loin dans le cours de seconde année.

Finalement, le formulaire est l'élément central de la base de données. L'utilisateur ne perçoit que lui, les requêtes (donc les tables) et le code VBA sous-jacent ne lui sont pas accessibles. Nous étudierons donc très finement les formulaires sur quatre séquences :

- celle-ci et les deux suivantes vous expliqueront comment réaliser des formulaires (aspect technique) ;
- la treizième vous indiquera comment réaliser des formulaires attractifs et corrects du point de vue de l'ergonomie (qualité de l'interface).

1B. Exemple complet

Vous vous souvenez sans doute de la base *Bibliothèque* que vous avez utilisée dans le cours de SQL. C'est une version simplifiée de la base que j'utilise chez moi (j'ai enlevé quelques champs sans objet pour le cours).

❶ *Access 2003 utilise la version 6.3 de Visual Basic. (La version précédente, Access 2000, utilisait la version 6.0. Il n'y a donc pas de changement majeur.)*

Voici les deux tables avec tous leurs champs. Je ne vous donne pas le MLD, mais la copie de la description des tables.

Livre : Table

Nom du champ	Type de données	Description
NumLivre	NuméroAuto	
NumAuteur	Numérique	
Titre	Texte	
TitreOriginal	Texte	
Langue	Texte	
Traducteur	Texte	
Cycle	Texte	
NumCycle	Numérique	
AnnéePublication	Numérique	Date de première publication (année seulement)
AnnéeImpression	Numérique	Date d'impression (année seulement)
Catégorie	Texte	Catégorie du livre (si poche)
Numéro	Numérique	
DateAchat	Date/Heure	
Prix	Monétaire	
NbrPages	Numérique	
NumCollection	Numérique	

Auteur : Table

Nom du champ	Type de données
NumAuteur	NuméroAuto
NomA	Texte
PrénomA	Texte
Initiales	Texte
Nationalité	Texte
DateNaissance	Date/Heure
LieuNaissance	Texte
DateDécès	Date/Heure
LieuDécès	Texte

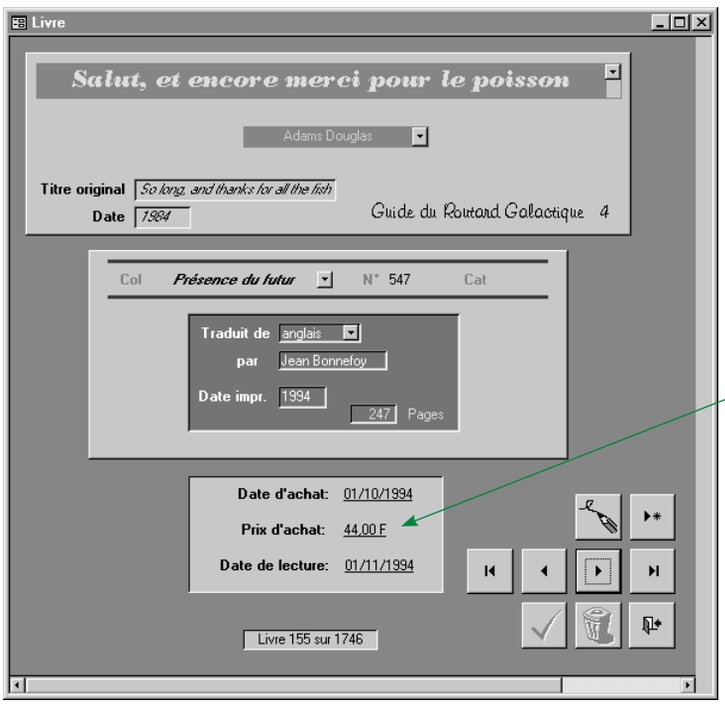
Comment entrer un livre dans la base ? Une solution consiste à ouvrir la table *Livre* en mode *Feuille de données* puis à rentrer les différentes informations.
Ci-dessous, j'ai fait une copie d'écran de la table *Livre* ouverte en *Feuille de données*. Je peux effectivement rentrer des livres :

Livre : Table

Num livre	Num c	Titre	Titre	Langue	Traducteur	Cycle	Num	Date pub.	Date i	Caté	Num.	Date d'achat	Prix	Pages	NumCollection
2323	777	Le voleur						1897	2003	F12	1798	23/05/2006	8,08 €	500	Folio classique
2324	778	Obermann						1804	1999	F14	1566	23/05/2006	9,03 €	530	Folio classique
2325	685	Moralités légendaires						1886	2000	F6	855	29/05/2006	4,66 €	243	Folio classique
2326	779	Kim	Kim	anglais	Louis Fabulet & (			2005	2005	F8	4206	29/05/2006	6,08 €	482	Folio classique
2327	61	En route						1895	2003	F11	2873	29/05/2006	7,60 €	648	Folio classique
2328	331	Tragédies complètes			Paul Mazon				2006	F8	360	29/05/2006	6,08 €	434	Folio classique
2329	780	Amphitryon - La comédi		latin	Pierre Grimal	Théâtre comple	1	2002	2002	F11	2308	29/05/2006	7,60 €	590	Folio classique
2330	780	Le soldat fanfaron - La c		latin	Pierre Grimal	Théâtre comple	2		2005	F11	2309	29/05/2006	7,60 €	520	Folio classique
2331	781	Énéide		latin	Jacques Perret				2005	F13	2225	15/06/2006	8,55 €	491	Folio classique
2332	52	Cellulaire	Cell	américair	William Olivier D			2006	2006			12/08/2006	22,00 €	407	Albin Michel
(éroAuto)															

Enr : 2322 sur 2322

Notez que tous les champs ne sont pas visibles dans la fenêtre. Il y en a trop ! De plus, pour associer un auteur au livre, il faut connaître son numéro. Bref, saisir un livre est très pénible.
La bonne solution consiste à utiliser un formulaire. Celui que je vous présente n'est pas d'une élégance extrême mais là n'est pas la question. L'important est de voir qu'il permet de saisir toutes les informations relatives à un livre.



*J'insiste, c'est un formulaire que j'ai réalisé étant jeune. Je ne possédais pas encore les notions d'ergonomie que je vous propose séquence 13. La preuve ? La présence d'un prix en **francs**, prouvant une réalisation antérieure à 2002.*

Ce formulaire permet donc de saisir toutes les informations relatives à un livre. Mais il apporte en plus trois choses intéressantes.

1. L'auteur (Douglas Adams) est identifié par son nom et non par son numéro. En pratique, la saisie du nom se fait par une liste déroulante qui ne propose que les auteurs stockés dans la base. Il est ainsi impossible d'avoir des données incohérentes. Mieux encore, l'utilisateur ne saisit pas un numéro très peu pratique mais le nom. Le formulaire, par contre, stockera le numéro d'auteur associé puisque la table *Livre* stocke le numéro et pas le nom. Nous y reviendrons.
2. En bas à droite du formulaire se trouve un ensemble de boutons de commande permettant de naviguer dans les enregistrements (livre précédent, suivant…), de modifier le livre courant, de valider ou d'annuler les modifications ou d'ajouter un livre. Lorsque vous cliquez sur l'un de ces boutons, vous exécutez une procédure VB. Il s'agit donc de programmation événementielle.
3. En bas du formulaire, vous avez une zone de texte indiquant de façon explicite le numéro du livre courant et le nombre total de livres dans la base.

Bref, le formulaire propose réellement des fonctionnalités intéressantes. L'affichage se veut convivial : c'est cela, l'ergonomie.

1C. Modes d'affichage

Tout formulaire possède par défaut la barre suivante :

Je vous rappelle qu'elle contient les *boutons de déplacement* permettant de changer l'enregistrement courant (sur lequel on est) de la source de données alimentant le formulaire. Dans l'ordre, on peut :

- se positionner sur le premier enregistrement ;
- passer à l'enregistrement précédent ;
- passer à l'enregistrement suivant ;
- se positionner sur le dernier enregistrement ;
- ajouter un nouvel enregistrement.

Identifiez ces actions sur la barre ci-dessus. Vous noterez que le formulaire de saisie des livres présenté dans le paragraphe précédent ne possède pas cette barre. C'est dû à un paramétrage explicite de ma part.
Il existe quatre modes d'affichage possibles pour un formulaire, chacun correspondant à un usage différent :

1. L'affichage en mode *Création*

C'est le mode permettant de créer le formulaire. Il vous concerne en tant qu'informaticien développant une application mais ne sera jamais exploité par l'utilisateur.

2. L'affichage *Formulaire unique*

Le formulaire est vu comme une fiche et ne présente qu'un seul enregistrement à la fois. Ces formulaires sont personnalisables (choix des contrôles, des couleurs et des formes, boutons, code…).

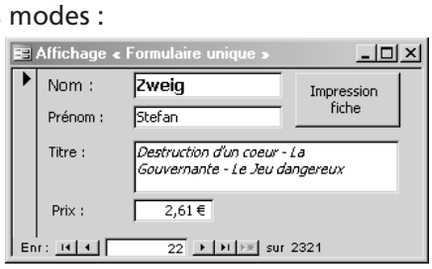
3. L'affichage en mode *Feuille de données*

C'est un affichage identique à celui de la table lorsqu'on l'ouvre pour ajouter des données. Les enregistrements sont présentés dans un tableau, les uns à la suite des autres. Cet affichage ne peut pas être travaillé pour améliorer son ergonomie.

4. L'affichage en mode *Formulaires continus*

C'est un mélange des deux modes précédents. Comme pour le mode *Feuille de données*, plusieurs enregistrements sont présentés à la fois. Cependant, le mode d'affichage est similaire au mode *Formulaire unique*. On peut donc le personnaliser.

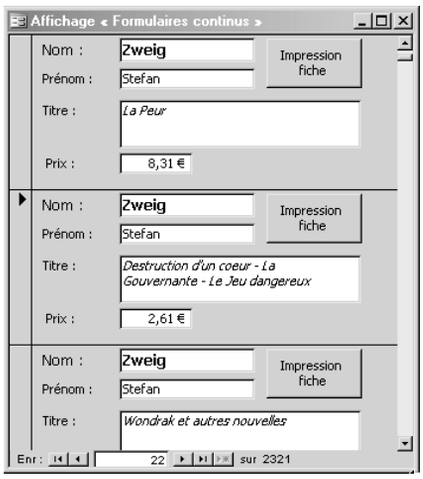
Voici un exemple complet. Regardez les formulaires ci-dessous sans chercher à comprendre comment je les ai réalisés : vous apprendrez cela plus tard dans cette séquence.
Sans toucher au paramétrage du formulaire, voici son aspect lorsqu'on l'affiche dans les différents modes :

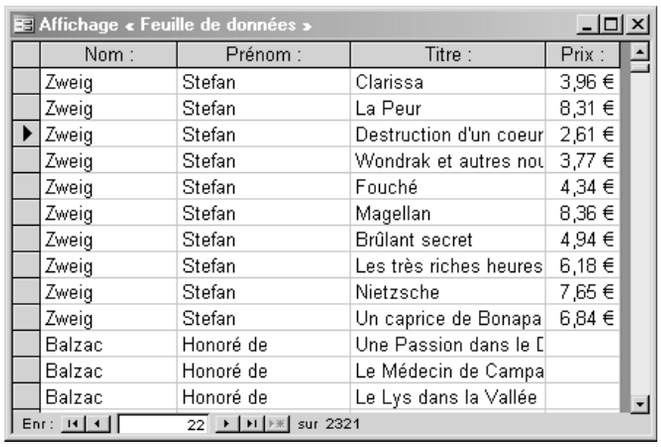


Le mode d'affichage Formulaire unique *est le plus utilisé.*

Avec ce mode, on n'affiche qu'un enregistrement à la fois.

Voici le même formulaire affiché dans les deux autres modes :





En affichage *Formulaires continus*, on voit plusieurs enregistrements à la fois. On a l'impression d'avoir plusieurs formulaires uniques les uns sous les autres : l'interface est la même. En retravaillant un peu le formulaire, voilà l'affichage que j'obtiens en mode *Formulaires continus* : un affichage style « feuille de données » avec une ergonomie de formulaire.

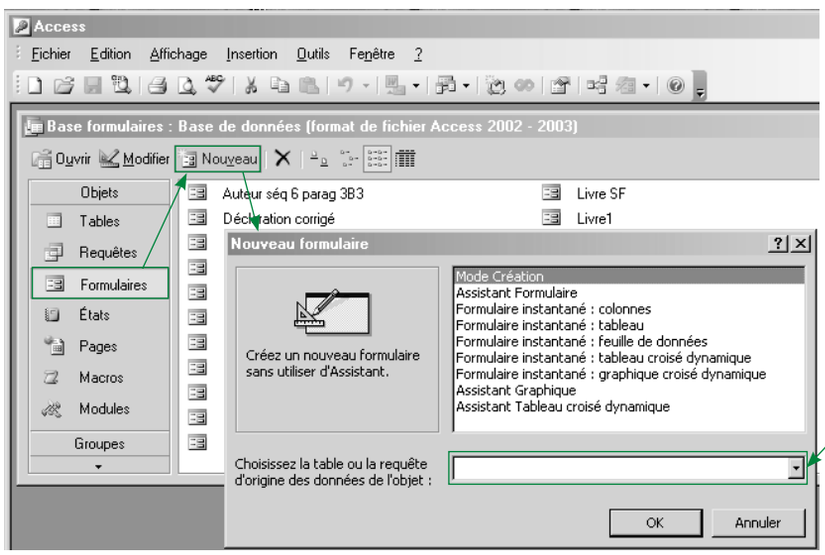
Notre formulaire de saisie des livres présenté dans le paragraphe 1B est affiché en mode *Formulaire unique* puisqu'il ne présente qu'un livre à la fois. Il est néanmoins beaucoup plus travaillé que celui ci-dessus. En fait, le formulaire précédent présenté dans les trois modes a été généré automatiquement par Access puis légèrement modifié (j'ai ajouté les mises en forme et le bouton).

Un formulaire généré par un assistant est très rapide à créer mais bénéficie d'une ergonomie limitée. Cela dit, rien n'empêche de la retravailler ! (Ce que j'ai légèrement fait.)

2. Création d'un formulaire

2A. *Les différents modes de création*

La création d'un formulaire se fait à partir de la fenêtre *Base de données*. Sélectionnez les objets *Formulaires* et cliquez sur le bouton *Nouveau*. Vous obtenez une fenêtre vous proposant divers modes de création.



Un formulaire est généralement basé sur une source de données (table ou requête jointure regroupant des informations issues de plusieurs tables).
Les champs affichés dans le formulaire seront un sous-ensemble de ceux de **la source de données.**

Explication des différents modes :

- le mode *Création* vous laisse tout faire (vous partez d'un formulaire vide) ;
- l'*Assistant Formulaire* vous pose des questions (quels champs de la source doit-on afficher, mode d'affichage, aspect du fond d'écran…) puis génère le formulaire correspondant ;
- les formulaires instantanés créent immédiatement un formulaire de l'aspect choisi en utilisant un paramétrage par défaut pour l'assistant ;
- les assistants *Graphique* et *Tableau croisé dynamique* parlent d'eux-mêmes.

Les trois formulaires précédents illustrant les modes d'affichage étaient des formulaires instantanés (donc vite réalisés !) auxquels j'ai rajouté un bouton. C'est une technique classique : on part d'un formulaire généré automatiquement et on l'adapte à nos besoins.

2B. *Source de données*

2B1. Qu'est-ce qu'une source de données ?

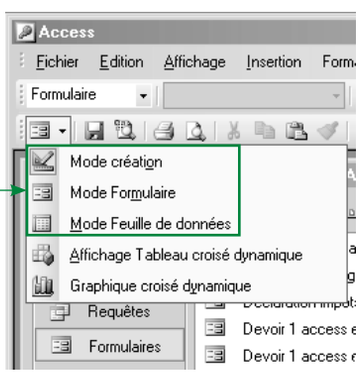
Je vous ai dit dans la séquence précédente que les formulaires et les états étaient basés sur une source de données (table ou requête).
Qu'est-ce qu'une source de données ? Et bien, de même qu'une source d'eau fournit de l'eau, une source de données fournit des données. Le formulaire (ou l'état) va donc puiser les informations qu'il affiche dans cette source exclusivement. Pour chaque formulaire, vous devrez donc écrire la requête renvoyant les champs et enregistrements que vous voulez voir figurer.
Plus précisément, la source contient des enregistrements. Il y a toujours un (et un seul) enregistrement à la fois qui est l'enregistrement courant (celui « sur lequel on est »).

Ce sont les valeurs des champs de cet enregistrement courant qui seront affichées par le formulaire. Vous pouvez donc les voir (évidemment) mais aussi les modifier. Pour visualiser ou modifier un autre enregistrement, il faut en faire l'enregistrement courant. Comment ? En utilisant la barre de déplacement (voir le paragraphe 1C) qui pilote la source de données.
Vous pouvez ne définir aucune source de données pour un formulaire. Dans ce cas, vous obtiendrez une gestion complètement personnalisée… à condition de la programmer. C'est ce que nous ferons l'année prochaine en option développeur d'applications.

2C. Changer de mode d'affichage

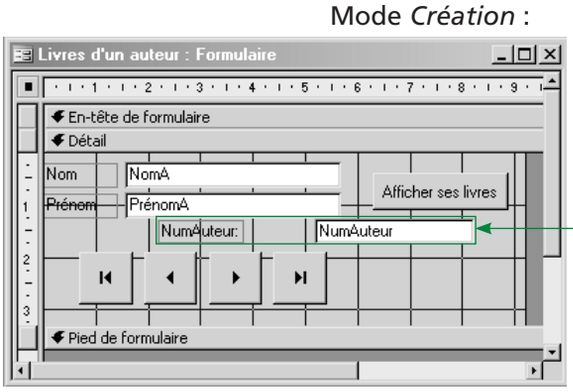
Le bouton bascule d'affichage disponible pour les tables et les requêtes est toujours là pour les formulaires (et sera encore là pour les états). *Affichage* est le premier bouton de la barre d'outils *Création de formulaire*.
Voici ses différents modes :

Les trois modes qui nous concernent sont :
1. *Le mode* Création
Il donne accès aux contrôles visuels et à la fenêtre Propriétés ; *nous étudierons ce mode et la création de formulaires dans le paragraphe 3.*
2. *Le mode* Formulaire
Cela correspond à son exécution. L'utilisateur manipule généralement les formulaires dans ce mode.
3. *Le mode* Feuille de données
Les données issues de la source de données sont affichées sous forme tabulaire.

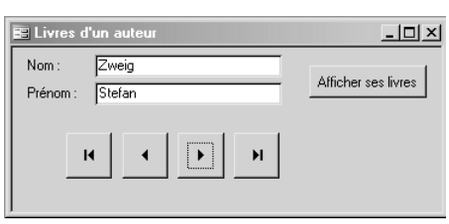


Illustrons cela en regardant le même formulaire dans les trois modes.

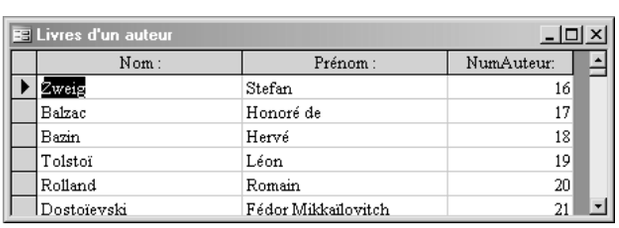
Mode *Création* :



Mode *Formulaire* :



Mode *Feuille de données* :



Si votre œil est attentif, vous avez dû remarquer que le champ *NumAuteur*, présent en mode *Création,* n'apparaît pas en mode *Formulaire*. Pourquoi ? Car je l'ai paramétré pour qu'il soit invisible. En effet, il me sert de variable locale : j'en ai besoin pour gérer

le formulaire mais il est inutile à l'affichage. Il est clair que la notion d'invisibilité n'existe pas en mode *Feuille de données* puisque l'on y voit ce champ.
Le mode *Création* affiche évidemment les contrôles invisibles puisque sinon on n'aurait aucun moyen d'y accéder (pour les rendre à nouveau visibles par exemple).

2D. Les différents formulaires instantanés

Le formulaire instantané peut être de type *Colonnes*, *Tableau* ou *Feuille de données*. Le formulaire est créé avec tous les champs de la source de données associée. Les différentes propriétés des champs définies lors de la création de la table (légende, règle de validité, valeur par défaut…) sont exploitées. Si ces valeurs n'ont pas été renseignées, il vous appartient de compléter manuellement les contrôles sur le formulaire. Idem si vous « mettez à jour » (c-à-d. « corrigez une erreur de conception en modifiant les tables ») la définition d'un champ dans une table.
Le formulaire instantané est le plus simple à réaliser car il est généré automatiquement par Access. L'avantage est qu'il est immédiatement opérationnel, l'inconvénient qu'il est rudimentaire. Mais rien ne vous empêche de partir d'un formulaire instantané puis de l'enrichir pour le transformer en formulaire de vos rêves.

Pour créer un formulaire instantané, vous devez procéder selon les étapes suivantes :

1. Cliquer sur les objets *Formulaires* de la fenêtre principale.
2. Cliquer sur le bouton *Nouveau.*
3. Choisir *formulaire instantané* (*Colonnes*, *Tableau* ou *Feuille de données*).
4. Choisir la source de données (table ou requête) sur laquelle est basé le formulaire.

Exercice

Dans la séquence 2, nous avons créé les tables *Livre* et *Auteur*. Reprenez cette base puis :

1. Créez des formulaires instantanés en colonnes pour les deux tables.
2. Utilisez-les ensuite pour visualiser le contenu des tables et modifier des valeurs de votre choix.
3. Vérifiez que les contrôles de validité définis lors de la création des champs sont toujours actifs.

Si les assistants instantanés fournissent un résultat globalement conforme à vos attentes mais que vous aimeriez quand même agir un peu sur le paramétrage, Access vous propose un assistant intéressant : l'*Assistant formulaire*. Ce dernier génère également le formulaire mais il vous pose quelques questions (choix de l'affichage, du format…) vous permettant de régler légèrement l'aspect final.

Exercice

Recommencez plusieurs fois la création des formulaires *Livre* et *Auteur* (en les nommant différemment bien entendu) en testant avec minutie les différentes options de l'*Assistant formulaire*.

2D1. Intérêt de ces assistants

L'ensemble du Pack Office possède de nombreux assistants qui font le travail à votre place. Cela est bien sympathique lors de tâches répétitives ou fastidieuses. Un autre rôle des assistants est de faire à votre place ce que vous ne savez pas faire. Enfin… cela ne vous concerne pas vu la formation que vous préparez ! Il s'agit de permettre à des non-informaticiens d'utiliser des assistants pour réaliser des choses qu'ils ne sauraient pas faire seuls. Répondre à des questions posées par un assistant (quelle couleur utiliser, quelle donnée afficher…) est à la portée de tout le monde. Paramétrer un formulaire demande déjà plus de connaissances : il faut accéder à la fenêtre des propriétés, savoir comment la remplir…

Word et Excel sont tout à fait abordables par de non-informaticiens car ce sont des outils bureautiques remplaçant les peu conviviales machines à écrire et à calculer. En revanche, Access est avant tout un outil de développement. Si vous ne connaissez rien à l'analyse ni à la programmation, vous serez cantonné aux assistants qui ne vous permettront pas d'aller très loin dans la conception.

En effet, Access nécessite des connaissances théoriques pour la mise en place des tables et pour l'automatisation des formulaires. Ce savoir n'est pas remplaçable par les assistants.

3. Mode *Création* (création ou modification de formulaires)

3A. Introduction

C'est ici que votre potentiel d'informaticien va se révéler. Nous allons voir comment créer un formulaire soi-même ou, ce qui revient au même, modifier un formulaire existant, qu'il ait été créé par un assistant ou par vous.
À quoi cela sert-il ? Vous allez pouvoir gérer l'ensemble du formulaire. Cela va de son contenu (quels champs afficher, comment les afficher) à l'aspect visuel (couleurs, formes…) en passant par son fonctionnement (zones en lecture seule, traitements automatisés…).

Je vous rappelle que nous étudierons la conception des formulaires dans cette séquence et les suivantes. Les aspects automatisation et ergonomie feront l'objet des séquences 12 et 13.

Notez que les formulaires Access sont tout à fait similaires aux feuilles du langage VB (et aux formulaires du langage Delphi). En fait, les contrôles et l'ergonomie ne sont pas propres à Access, mais à l'ensemble des outils de développement visuels (graphiques) constituant la norme actuelle.

D'ailleurs, les contrôles visuels que nous allons étudier (boutons, zones de liste…) vous sont familiers puisqu'ils proviennent de l'environnement Windows. La seule nouveauté est que vous allez apprendre à intégrer ces contrôles dans vos applications.

Pour créer ou modifier un formulaire, il faut passer en mode *Création*. Si vous avez un formulaire affiché, utilisez le bouton d'affichage vu dans le paragraphe 2C. Sinon, utilisez les boutons *Nouveau* ou *Modifier* de la fenêtre principale (une fois les objets *Formulaires* sélectionnés).
Pour étudier l'ensemble d'un formulaire, nous travaillerons sur un exemple complet. Nous allons réaliser un formulaire permettant de saisir les auteurs dans la base *Bibliothèque*.

Je vous rappelle le MLD :

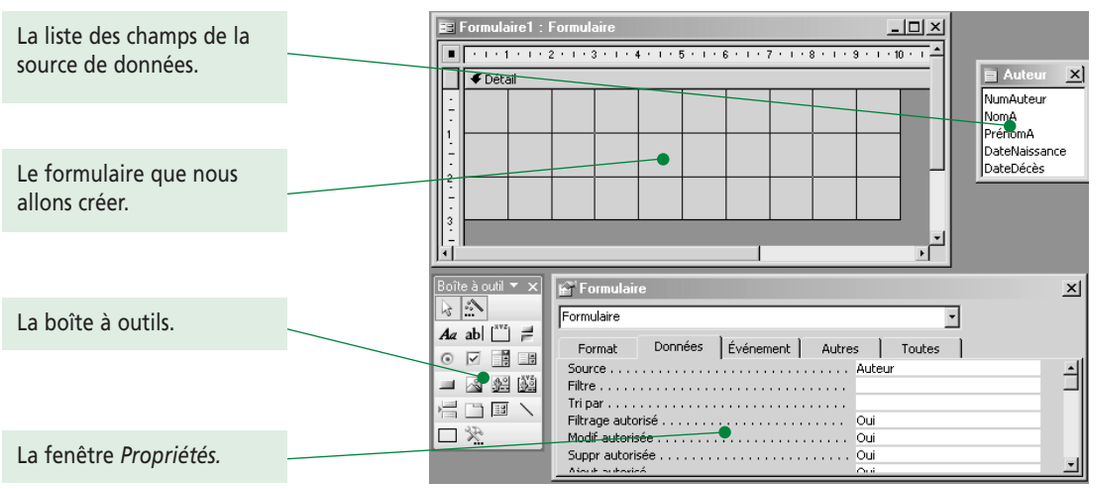
Livre (NumLivre, Titre, NbrPages, DateAchat, PrixAchat, NumAuteur#)
Auteur (NumAuteur, NomA, PrénomA, DateNaissance, LieuNaissance)

Légende : clé primaire, clé étrangère#

Nous avions fait un formulaire instantané dans le paragraphe 2D. Cette fois, foin d'assistant !

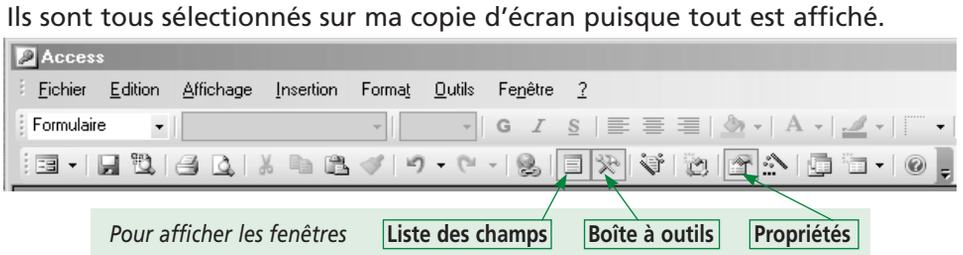
3B. L'interface de travail sur un formulaire

Partez de la fenêtre *Base de données* (*F11* pour l'afficher), sélectionnez les objets *Formulaires* et cliquez *Nouveau*. Choisissez le mode *Création* et indiquez la table *Auteur* en source de données. Validez… Vous obtenez alors un écran contenant quatre éléments distincts (peut-être présentés différemment, voir ci-dessous) :



Nous allons étudier ces différents éléments. Encore une fois, ils sont sans doute présentés différemment chez vous (voire absents), notamment la boîte à outils qui est sûrement une barre verticale sur la gauche de votre écran. Comme toute barre d'outils du *Pack Office*, vous pouvez la déplacer comme vous le souhaitez.

Voici les boutons d'affichage correspondant dans la barre d'outils *Création de formulaire*. Ils sont tous sélectionnés sur ma copie d'écran puisque tout est affiché.



S'il vous manque l'une des fenêtres, vous savez maintenant comment l'afficher. Évidemment, je ne le détaille plus, mais vous trouvez également ces commandes dans le menu *Affichage*. Observez d'ailleurs le raccourci clavier pour la fenêtre des propriétés❶. Ce sera notre fenêtre de travail principale, donc il est de bon ton❷ de savoir y accéder très rapidement.

❶ *Je vous aide, c'est* Alt+Enter. *(C'est marqué à droite de la commande* Propriétés *dans le menu* Affichage.*)*

❷ *« Ça fait bien ».*

3C. La fenêtre Propriétés et les propriétés

Si elle n'est pas affichée sur votre écran, cliquez sur le bouton correspondant ou menu *Affichage/Propriétés* ou *Alt+Entrée*.

C'est une fenêtre essentielle à tout langage de développement visuel (Visual C++, VB, Delphi, Access…). Elle permet de définir les caractéristiques des formulaires ou autres. C'est un peu le principe des propriétés des champs vues dans la séquence 3.
Imaginez une voiture d'une marque et d'un modèle donné. Par exemple, prenons une Peugeot 205. Ces voitures possèdent différentes caractéristiques : leur couleur, leur puissance, le type de boîte (automatique ou manuelle), la motorisation (diesel ou essence), les équipements (climatisation, auto-radio…), etc. Il y a donc une grande variété de 205 en fonction de ces différentes caractéristiques.

On peut aller plus loin en n'envisageant plus la 205 mais une voiture en toute généralité. À toutes les caractéristiques précédentes (couleur, puissance, boîte…), je rajoute alors les caractéristiques *Marque* et *Modèle*.
Quel est l'intérêt de cela ? Et bien, lorsque je veux acheter une voiture, je pars du concept de voiture que tout le monde connaît. Je n'ai donc pas à expliquer ce que j'entends par *voiture*. Ensuite, je n'ai qu'à détailler les caractéristiques que je souhaite. Par exemple : marque *Peugeot*, modèle *205*, couleur *indifférente*, âge *entre 3 et 5* (je veux donc une voiture d'occasion)…

De la même façon, un ordinateur (en toute généralité) sera décrit par les caractéristiques suivantes : type et fréquence du processeur, type et capacité RAM, type de disque dur… Notez que certains constituants de l'ordinateur peuvent eux-mêmes posséder des caractéristiques. Si le format du boîtier est juste une valeur (*desktop*, *mini-tour*), le disque dur est un objet en tant que tel défini par les propriétés *marque*, *capacité*, *vitesse de rotation*…
Vous ne devez pas être trop perdu car on retrouve les notions d'entité et de propriété vues dans le cours d'analyse. Ce qui me permet de vous citer encore ma grande idée : l'informatique n'invente rien, elle se contente d'exploiter les concepts du monde réel.
Revenons à nos moutons Access : quel est l'intérêt de cette digression ? Et bien, les outils de développement modernes exploitent ce mécanisme intuitif d'objets et de caractéristiques. L'idée est la suivante : tout ce que vous manipulez dans les outils de développement visuel est un objet ayant des caractéristiques appelées *propriétés*. C'est à vous de définir les propriétés pour que l'objet soit conforme à vos désirs.

Prenons deux exemples :

- l'objet *Formulaire* est caractérisé par sa couleur de fond (grise par défaut), le fait que les boutons de déplacement soient ou non présents… Cela donnera autant de propriétés ;
- l'objet *Bouton* possède une couleur, des dimensions, un texte affiché, du code qui sera exécuté lorsque l'on cliquera dessus…

Je viens de consacrer ce dernier quart d'heure à compter les propriétés de l'objet *Formulaire*. À une ou deux près, car ces chiffres me donnent un peu le tournis, on dispose de :

- 107 propriétés accessibles par la fenêtre *Propriétés*. Elles permettent les paramétrages visuels, des données et des événements (code réagissant aux interactions) ;
- 283 propriétés au total, les 176 restantes n'étant visibles que par programmation.

J'ajoute qu'un formulaire est constitué de différentes sections (en-tête…) possédant chacune des propriétés. Cela doit-il vous inquiéter ? Non, car plus il y a de propriétés, plus vous pouvez paramétrer finement votre formulaire si vous le souhaitez. Il est inenvisageable de les connaître par cœur, voire de les connaître tout court. Lorsque vous voulez changer un réglage, vous parcourez les propriétés ou l'aide pour trouver celle qu'il faut modifier.
En fait, seules quelques dizaines de propriétés sont d'un usage courant. Et encore, beaucoup ont des valeurs prédéfinies qu'il n'est pas utile de changer. Enfin, de nombreuses propriétés se retrouvent sur différents objets. Par exemple, tout contrôle possède un nom, une couleur…
Le principe de création d'un formulaire (ou d'un état) est le suivant :

- vous partez d'un formulaire dont vous paramétrez les propriétés (quelle couleur, quelle taille…) ;
- vous ajoutez les contrôles que vous voulez (zone de texte, boutons…) et vous paramétrez leurs propriétés ;
- c'est fini !

Bien entendu, c'est plus facile à dire et lire qu'à faire. Je termine sur la fenêtre *Propriétés* en précisant que :

- pour une manipulation plus aisée, les propriétés sont regroupées par onglets thématiques (*Format*, *Données*…) et sont toutes récapitulées dans l'onglet *Toutes* ;
- la fenêtre *Propriétés* affiche les propriétés de l'objet courant (le titre de la fenêtre est le nom de l'objet sélectionné) : si vous cliquez sur un formulaire, ce sont ses propriétés qui s'affichent ; cliquer sur un bouton affichera les siennes. Autrement dit, pour afficher (et modifier) les propriétés d'un objet, cliquez dessus !
- quand vous cliquez dans la zone de texte d'une propriété pour la modifier, vous pouvez lancer l'aide par *F1* et obtenir des explications précieuses sur la propriété en cours (à quoi elle sert, quelle valeur mettre…).

3D. Le formulaire

3D1. Présentation

Il est certain que vous avez au moins cette fenêtre. C'est le formulaire que nous allons créer. Pour le moment, il est petit et gris ! Le formulaire possède deux modes essentiels :

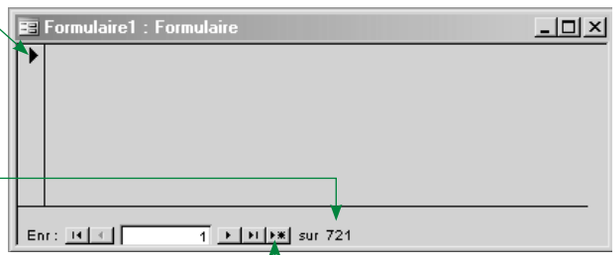
- le mode *Création* qui permet de le créer et de le modifier ;
- le mode *Formulaire* qui permet de l'utiliser.

En faisant le parallèle avec la programmation, le mode *Création* correspond au source et le mode *Formulaire* à l'exécution.
Nous sommes actuellement en mode *Création*. Le quadrillage n'apparaît que dans ce mode pour vous aider à aligner les contrôles. Passez au mode *Formulaire* (je le répète une dernière fois, cliquez sur le bouton ou menu *Affichage/Mode formulaire*).

Voici ce que l'on obtient :

C'est le même triangle que dans les tables. Il indique l'enregistrement courant. Si vous modifiez les champs (ici, il n'y en a pas), il se transforme en crayon.

Comme indiqué dans le paragraphe 1C, les boutons de déplacement correspondent à la source de données. Ici, c'est le premier auteur qui est affiché (j'en ai **721** dans ma base). Vous noterez que pour ajouter un nouvel enregistrement, on clique sur le bouton ▸*.



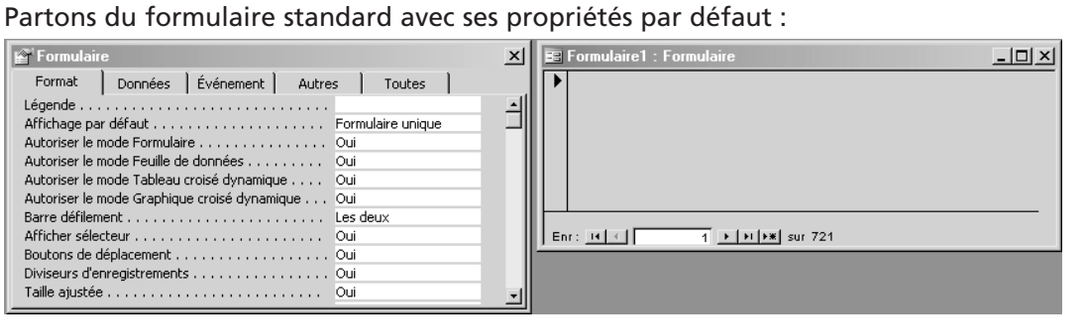
Je viens de dire que c'était le premier auteur qui était affiché et pourtant, il n'y a strictement rien dans le formulaire. Manipulez les différents boutons pour naviguer dans les enregistrements, voire cliquez sur le bouton permettant d'en rajouter un : il ne se passe rien. C'est normal, nous n'avons rien mis dans le formulaire ! Comprenez bien la chose suivante : dans la source de données, vous êtes actuellement positionné sur le premier enregistrement de la table *Auteur*. Les valeurs des champs affichés seront donc les valeurs de ce premier enregistrement. Et quels champs de la source affiche-t-on ? Pour le moment aucun.

Imaginez-vous dans un supermarché. C'est la source de données. Le rayon où vous êtes, c'est l'enregistrement courant. Les produits à votre disposition, ce sont les champs de l'enregistrement. Votre liste de courses représente les champs affichés dans le formulaire. Une liste vide entraînera un caddie vide même si le supermarché est plein. (Bon, ma métaphore est assez approximative, mais nous ferons avec.)

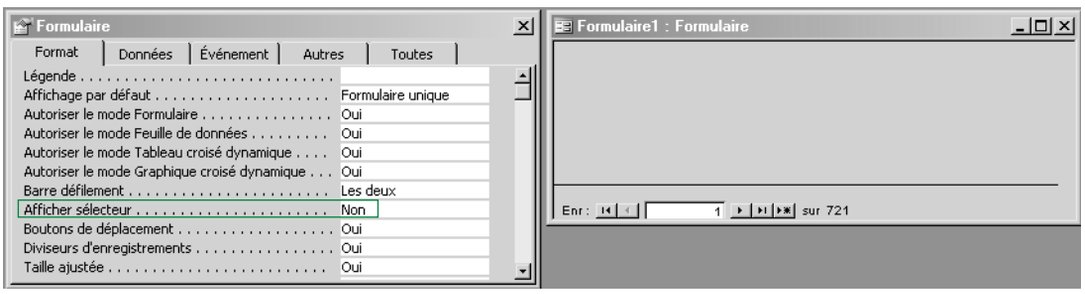
Même lorsque le formulaire est en mode *Affichage*, on peut changer certaines propriétés. Cela permet de réaliser des modifications et d'observer en direct le résultat. Attention, certaines propriétés doivent être définies en mode *Création* : leur modification en mode *Affichage* n'aura aucun effet. (On doit alors basculer de *Affichage* en *Création*, faire les modifications puis revenir en *Affichage* pour voir le résultat.)

3D2. Quelques propriétés du formulaire

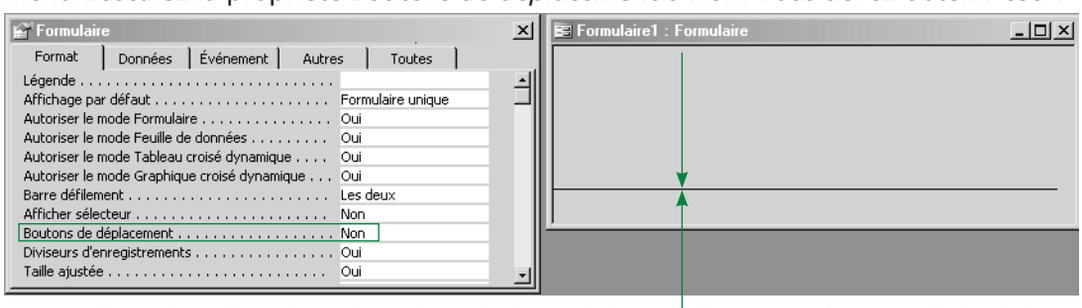
Partons du formulaire standard avec ses propriétés par défaut :



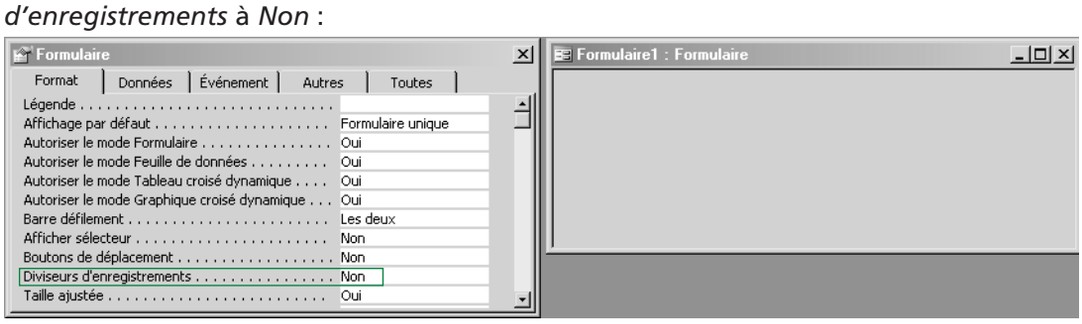
Nous allons supprimer la barre rectangulaire à gauche contenant le triangle. Le formulaire étant activé (mode *Formulaire*), allez dans la fenêtre *Propriétés*, onglet *Format*. Basculez la propriété *Afficher sélecteur* à *Non*, soit en utilisant la liste modifiable, soit en double-cliquant sur la valeur. Lorsque vous faites la modification, observez que le formulaire change comme illustré ci-après.



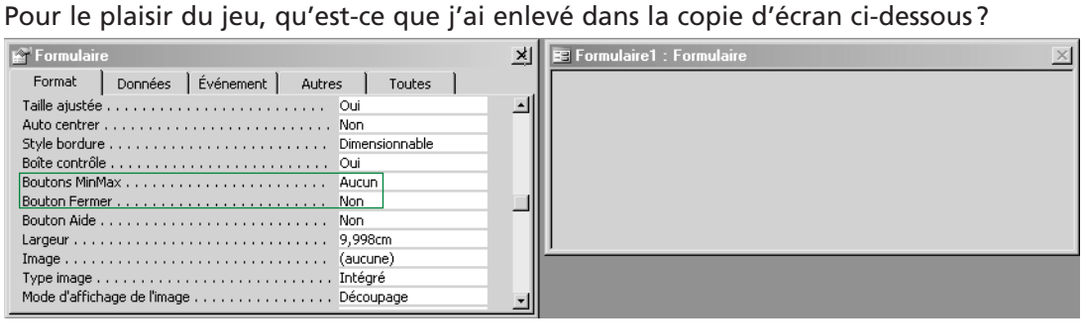
Comme, c'est marrant, nous continuons en rafale en supprimant les boutons de déplacement. Basculez la propriété *Boutons de déplacement* à *Non*. Vous devez obtenir ceci :



Pour enlever l'espèce de trait en bas du formulaire (**ceci**), ce sera la propriété *Diviseurs d'enregistrements* à *Non* :

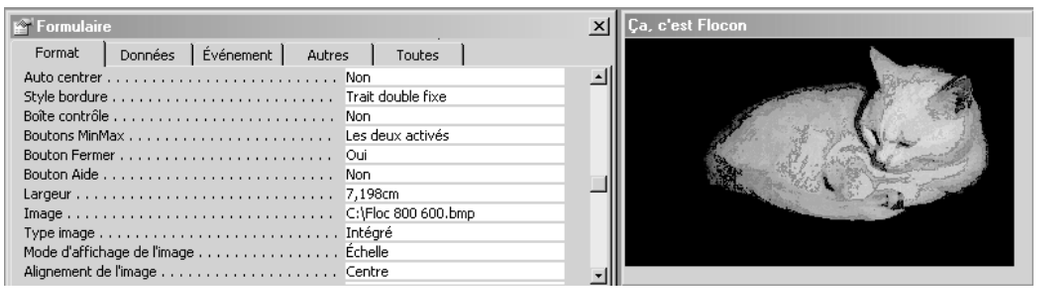


Pour le plaisir du jeu, qu'est-ce que j'ai enlevé dans la copie d'écran ci-dessous ?



Et là, je n'aurais pas discrètement placé mes teckels ?

C'est une vraie drogue. Allez, encore un coup et j'arrête.



Tout cela pour vous montrer qu'en quelques secondes, on peut radicalement changer l'aspect du formulaire en modifiant très simplement son paramétrage.

Exercice

Testez les autres propriétés ayant un effet visuel. Faites cet exercice avec soin pour découvrir tout ce qui peut être fait.

3D3. Explication sur quelques propriétés

Revenons au cours. Vous remarquerez qu'en enlevant les boutons de déplacement, je ne peux plus me déplacer dans les enregistrements ni en rajouter puisque le bouton d'ajout et les autres ont disparu. Quel est l'intérêt d'un formulaire sans ces boutons ? Deux cas se présentent :

- le formulaire peut n'être basé sur aucune source de données car il sert à effectuer des paramétrages. Un exemple ? Et bien, lancez le menu *Fichier/Enregistrer sous…* La fenêtre que vous obtenez est un formulaire sans rapport avec une source de données (on parle de boîte de dialogue) ;
- si le formulaire est basé sur une source de données, il faut d'une manière ou d'une autre permettre de naviguer dans les enregistrements ou d'en rajouter. Comment faire ? Tout simplement en ajoutant des boutons dans le formulaire qui exécuteront du code VB réalisant le déplacement ou l'ajout. Pourquoi se fatiguer à ajouter des boutons plutôt qu'utiliser ceux qui sont déjà là ? Car ces derniers sont petits et pas forcément explicites pour un non-informaticien. Pour des raisons d'ergonomie, on peut vouloir les remplacer par ses propres boutons (revoyez le formulaire du paragraphe 1B). Nous verrons cela dans la séquence 12 sur la programmation.

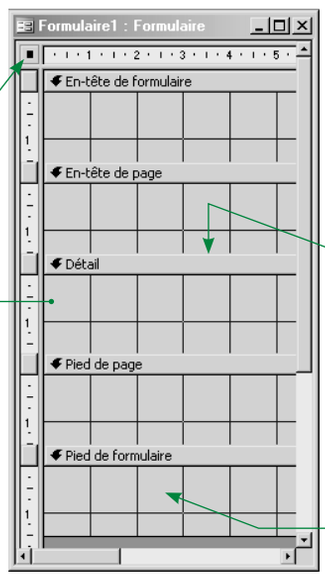
Dans l'onglet *Données*, vous remarquerez la propriété *Source* : elle contient *Auteur* ainsi que nous l'avons indiqué dans la fenêtre de création du formulaire. Vous pouvez choisir une autre source de données (évidemment, les contrôles déjà posés sur le formulaire seront alors à revoir).
La barre de titre indique un nom par défaut peu parlant (*Formulaire1*). Dans l'onglet *Format*, propriété *Légende*, vous pouvez taper un autre titre. Essayez et observez la mise à jour.
Pour toutes les propriétés, pensez à consulter **l'aide en ligne** !

3D4. Les différentes sections du formulaire

Revenez au mode *Création*. Un formulaire est composé des cinq sections : un en-tête et un pied de formulaire❶, un en-tête et un pied de page❷ et une zone détail.

En affichant toutes les sections, on obtient :

Pour afficher les propriétés du formulaire, il faut le sélectionner. Pour cela, il suffit de cliquer **ici**. *(La pastille noire indique que c'est le formulaire dans son ensemble qui est actuellement sélectionné.)*



Pour changer la taille d'une section, cliquez sur son bord inférieur et glissez. Par exemple, cliquez **ici** *et glissez pour modifier la taille de la section* En-tête de page.

La section **Détail** est la plus importante. Elle contiendra le détail de ce qui doit être affiché pour chaque enregistrement. Nous y reviendrons évidemment.

Pour accéder aux propriétés d'une section, il faut la sélectionner en cliquant dedans. Par exemple, pour sélectionner la section Pied de formulaire, *c'est* **là** *qu'il faut cliquer.*

Sélectionnez maintenant la zone *Détail* du formulaire en cliquant dedans. Vous remarquerez que la fenêtre *Propriétés* change : son titre devient *Section : Détail* et les propriétés ne sont plus les mêmes. Dans l'onglet *Format*, testez *Couleur fond* :

- tapez une valeur ou cliquez sur le bouton [...] pour avoir accès à la palette ;
- pour valider la modification, il faut sortir de la zone de saisie ;
- restaurez ensuite la sobre couleur initiale.

Ce petit tour des propriétés est sensé vous avoir convaincu de la simplicité du paramétrage des objets, ici le formulaire. Prenez le temps de regarder, tester et lancer l'aide sur les différentes propriétés du formulaire et de la zone *Détail*. Ce n'est que comme cela que vous saurez exactement tout ce que vous pouvez faire !

L'en-tête et le pied de formulaire sont des concepts connus depuis Word. Vous pouvez y mettre un titre, la date ou toute autre chose. Enfin… si cela a éventuellement un intérêt sous Word (et encore, je vous déconseille de trop les remplir), leur usage est plus réduit dans un formulaire :

- mettre un titre ? Ma foi, c'est le rôle de la barre de titre de la fenêtre ;
- mettre la date ? C'est inutile car le formulaire est affiché le jour où on le consulte !

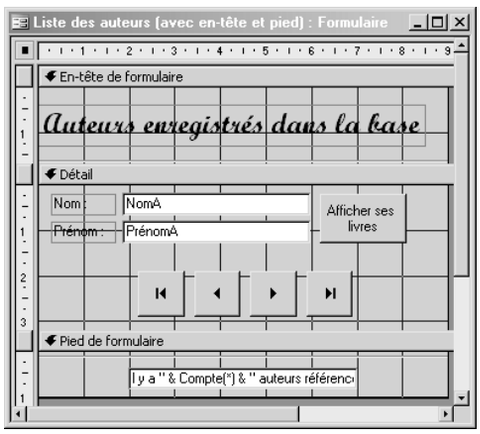
Les en-têtes et pieds de pages n'étant destinés qu'à l'impression, je les cache à partir de maintenant car je n'en ai pas besoin.

Et la zone *Détail* ? En fait, vous la connaissez également sous Word puisqu'elle correspond à la page elle-même (bref, à ce qui est entre l'en-tête et le pied). Dans un formulaire, elle contient les enregistrements affichés.

❶ *Pour les afficher, c'est la commande* Affichage/En-tête/pied de formulaire. *Ces deux sections sont cachées par défaut car elles servent rarement.*

❷ *Pour les afficher, c'est la commande* Affichage/En-tête et pied de page. *Ces deux sections sont cachées par défaut. Vous devinez pourquoi ? C'est parce qu'elles servent rarement.*

Voici un formulaire complet (avec les trois zones remplies) dans les différents modes :



***Voici le mode* Création**
J'ai placé :
- *un titre et un trait dans l'en-tête ;*
- *un trait et une expression dans le pied. Cette dernière m'indiquera combien j'ai d'auteurs dans ma base. (Nous reverrons cela.)*

La zone Détail *me permet de décrire l'affichage d'un enregistrement. Cette description (quels contrôles, taille, couleur, boutons...) ne sera utilisée que dans le mode* Formulaire.

***Voici le mode* Formulaire unique**
On voit l'en-tête et le pied.
Les contrôles (zones de texte et boutons) sont affichés comme je l'ai demandé dans le mode Création.

***Voici le mode* Feuille de données**
On remarque que toute la mise en forme est perdue :
- *en-tête et pied de page ne sont pas affichés ;*
- *je n'ai plus de bouton ;*
- *les deux champs* Nom *et* Prénom *sont toujours affichés mais sur une ligne.*

En fait, cet affichage est identique à celui des tables.

J'insiste sur le contenu de la zone *Détail*. On y décrit le style d'affichage de l'enregistrement courant. Il est donc tout à fait impossible de décrire deux enregistrements… Voyez-vous où je veux en venir ? Et bien, tous les enregistrements seront affichés de la même façon. Vous ne pourrez pas afficher d'abord le nom ou d'abord le prénom selon l'enregistrement à moins de tout programmer. Est-ce grave ? Non ! Ce genre d'affichage variable est de toute façon à proscrire (voir la séquence 13 sur l'ergonomie).

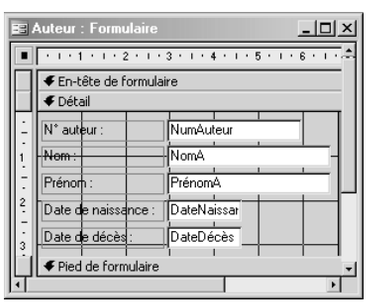
3E. Le contrôle

Menu *Affichage/Liste des champs* ou bouton de la barre d'outils *Création de formulaire* si elle n'est pas affichée. Cette petite fenêtre (redimensionnable si tous les champs n'apparaissent pas) contient les champs de la source de données. D'ailleurs, sélectionnez le formulaire et changez la propriété *Source* (onglet *Données* dans la fenêtre *Propriétés*) en *Livre*. Observez la mise à jour de la fenêtre *Liste des champs*. Remettez la source *Auteur*.

À quoi sert cette fenêtre ? Elle vous permet de placer rapidement les champs disponibles sur le formulaire. C'est très simple : vous cliquez (et maintenez) sur un champ de la liste et vous le glissez sur le formulaire (dans la zone *Détail*). Vous lâchez… Access crée un contrôle affichant le champ correspondant. Vous pouvez sélectionner plusieurs champs dans la fenêtre (avec la touche *Ctrl* ou *Maj*) et les glisser d'un coup !

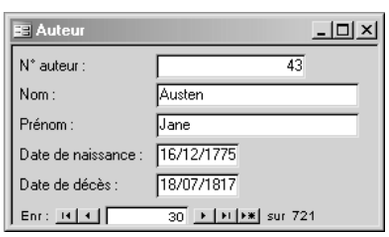
En quelques manipulations de souris, je remplis le formulaire. Ci-dessous, j'ai fait une copie écran du formulaire en mode *Création* et deux copies en mode *Formulaire* présentant des enregistrements différents de la source de données.

1re copie : mode *Création*



2e copie : mode *Formulaire*

3e copie : mode *Formulaire*



Vous noterez que :

- j'ai remis les boutons de déplacement. Ils permettent de voir que dans la 2e copie écran je suis positionné sur le 3e enregistrement de la source de données (la valeur de sa clé primaire est 16) et dans la 3e copie, je suis sur le 30e (pour lequel *NumAuteur* vaut 43). Je vous rappelle que la valeur de la clé primaire (ici *NumAuteur*) n'a rien à voir avec le numéro (position) de l'enregistrement dans la source de données ;
- chaque contrôle est constitué de deux parties :
  1. Une **étiquette** (du texte noir sur fond transparent) permettant d'identifier la saisie❶.
  2. Une **zone de saisie** (fond blanc❷) permettant la saisie de la valeur du champ. En mode *Création*, la zone de saisie contient le nom du champ auquel elle est associée. En mode *Formulaire*, elle contient la valeur du champ de l'enregistrement courant de la source de données.

  Prenons comme exemple le deuxième contrôle, qui permet de saisir le nom de l'auteur. Son étiquette est « *Nom :* ». En mode *Création*, la zone de saisie affiche « *NomA* » ; cela signifie qu'elle est reliée au champ *NomA* de la source de données comme illustré ci-après.

❶ *D'où viennent les valeurs des étiquettes ? Nous l'avons déjà vu : ce sera la légende du champ si elle a été définie lors de la création de ce dernier. À défaut de légende, ce sera le nom du champ. Par exemple, le champ* LieuNaissance *a pour légende « Lieu de naissance » ; c'est elle qui a été utilisée. Comme le champ* NumAuteur *n'en a pas, c'est son nom qui sert d'étiquette. Cet exemple doit vous inciter à utiliser les légendes pour améliorer l'ergonomie !*

❷ *Toutes les caractéristiques d'affichage sont modifiables par les propriétés.*

Mode *Création*

En mode *Formulaire*, le formulaire affiche les valeurs des champs de l'enregistrement courant de la source de données. Dans la 2e copie écran ci-dessus, l'enregistrement courant est le troisième de la source de données. La valeur du champ *NomA* est *Zweig*. C'est donc cette valeur qui est affichée. Dans la 3e copie, je suis sur le 30e enregistrement qui est celui de l'auteur J. Austen. Ci-dessous, voici, sur un autre enregistrement, la zone de texte associée au nom en mode *Formulaire* :

Mode *Formulaire*

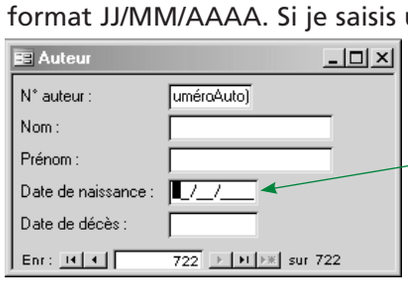
• tous les champs disponibles dans la source de données n'ont pas été utilisés pour générer des contrôles. Cela n'est absolument pas une obligation. D'ailleurs, dans un vrai formulaire, on ne met généralement pas le numéro identifiant car c'est une valeur technique qui a un sens pour l'application mais pas pour l'utilisateur❶. Bien entendu, si un champ ne dispose pas de contrôle pour l'afficher dans le formulaire, l'utilisateur ne pourra ni le voir ni le modifier… ce qui est très bien pour l'identifiant !

Exercice

Vérifiez que vous distinguez bien les différentes notions *champ*, *contrôle*, *source de données* et *enregistrement*. Si ce n'est pas le cas, les remarques précédentes n'ont pas dû vous sembler claires. Revoyez alors un peu le cours, faites quelques manipulations, consultez l'aide, bref, expérimentez !

3F. Contrôles liés et propriétés des champs

Dans les copies ci-dessus, vous observez que les dates de naissance et décès sont affichées au format JJ/MM/AAAA. Si je saisis un nouvel auteur, que se passe-t-il ?



Et bien, lorsque je saisis une date, j'obtiens un masque de saisie « __/__/____ ». Et pas moyen de saisir une lettre !

C'est particulièrement pratique, mais je m'interroge car je n'ai rien paramétré de particulier dans le formulaire. En fait, si je n'ai rien fait dans le formulaire, j'ai défini les propriétés fines du champ lors de la création de la table.

❶ *Attention, ne me dites pas que vous devez savoir que Jane Austen est l'auteure 43 afin d'associer ses livres à l'auteur 43 (en mettant 43 dans la clé étrangère* NumAuteur de Livre). *En effet, une ergonomie correcte vous fera utiliser une liste déroulante affichant les noms des auteurs, sachant qu'en sous-main, Access stockera le numéro associé. Nous reverrons évidemment cela, c'est très important. [Je dis souvent cela dans ce cours ; mais je ne mens pas, en fait, tout est très important.]*

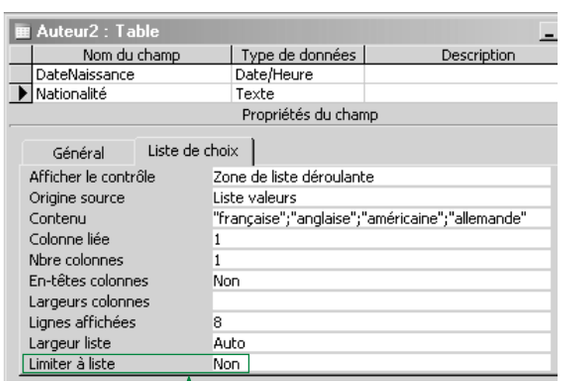
Reprenons cela : comme je vous l'avais dit, les différentes caractéristiques des champs sont exploitées pour paramétrer le contrôle visuel affiché. En pratique, il se passe deux choses lorsque vous faites glisser un champ dans le formulaire :

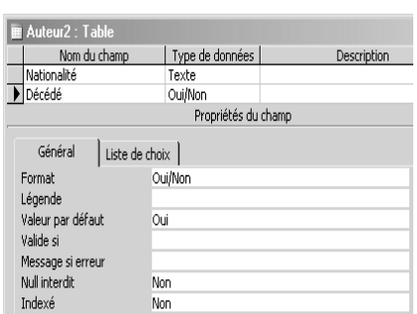
- Access choisit parmi tous les contrôles disponibles le plus adapté au type du champ. Par exemple, une valeur booléenne (de type *Oui/Non*) sera représentée par une case à cocher ;
- les différentes propriétés définies pour le champ (valeur par défaut, masque de saisie, légende…) sont prises en compte pour le paramétrage du contrôle visuel. En effet, un contrôle est un objet comme un autre : à l'instar des formulaires, il possède des propriétés permettant de le personnaliser. Si vous n'avez pas défini les propriétés pour les champs, vous devrez définir à la main les propriétés des contrôles.

Nous allons illustrer ces deux points en créant une table *Auteur2*, copie de la table *Auteur* à deux bémols près :

- je supprime le champ *DateDécès* et je le remplace par un champ *Décédé* de type *Oui/Non*. Comme j'ai plutôt dans ma base des auteurs anciens, je mets comme valeur par défaut *Oui* ;
- la majorité de mes auteurs sont français, anglais, américains ou allemands. Je vais donc utiliser une zone de liste déroulante pour la saisie de la nationalité. Comment faire ? Soit vous la définissez à la main (onglet *Liste de choix* à côté de *Général* et vous copiez les propriétés ci-dessous) soit vous utilisez le type de données *Assistant Liste de choix…* qui vous posera des questions et paramétrera la liste pour vous. (Pour plus d'explications, lire la suite et l'aide en ligne !)

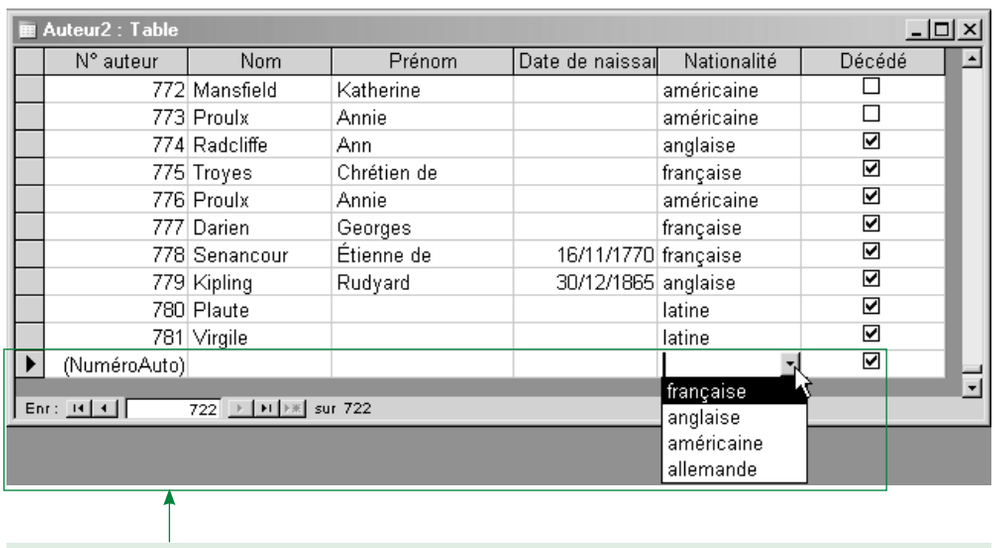
Voici des copies écran des propriétés des champs *Nationalité* et *Décédé* :





Notez **ce paramétrage :** *dans ma liste, je n'ai mis que les quatre nationalités les plus fréquentes pour alléger l'interface. Le fait que la saisie ne soit pas limitée à la liste permet à l'utilisateur soit de saisir l'une des valeurs de la liste, soit de taper une valeur différente.*
Limiter à la liste est parfois une bonne idée, parfois non. Cela dépend du champ, il faut se poser la question à chaque fois.

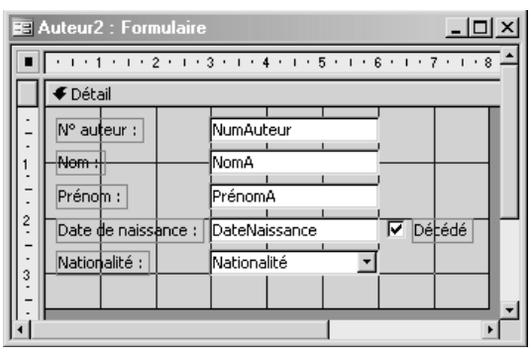
Lorsque j'affiche la table en mode *Feuille de données*, ces réglages sont pris en compte comme nous l'avons vu lors des définitions des tables.

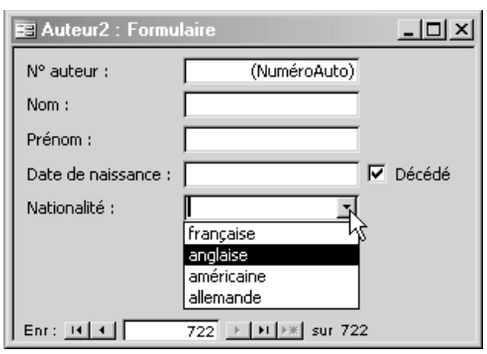


Vous remarquerez que :

- *la saisie du champ* Décédé *est faite par une case à cocher : vous n'avez pas à entrer* oui *ou* non *au clavier ;*
- *lors de la saisie d'un nouvel auteur,* Décédé *est coché. La valeur par défaut est bien prise en compte ;*
- *la saisie de la nationalité se fait à partir d'une liste déroulante. Elle possède une propriété* Limité à liste *qui vaut* Non *par défaut. Dans ce cas, l'utilisateur peut saisir une valeur ne faisant pas partie de la liste. C'est bien entendu indispensable ici puisque je me suis limité aux quatre nationalités les plus fréquentes. Par exemple, les deux derniers auteurs saisis sont d'expression latine.*

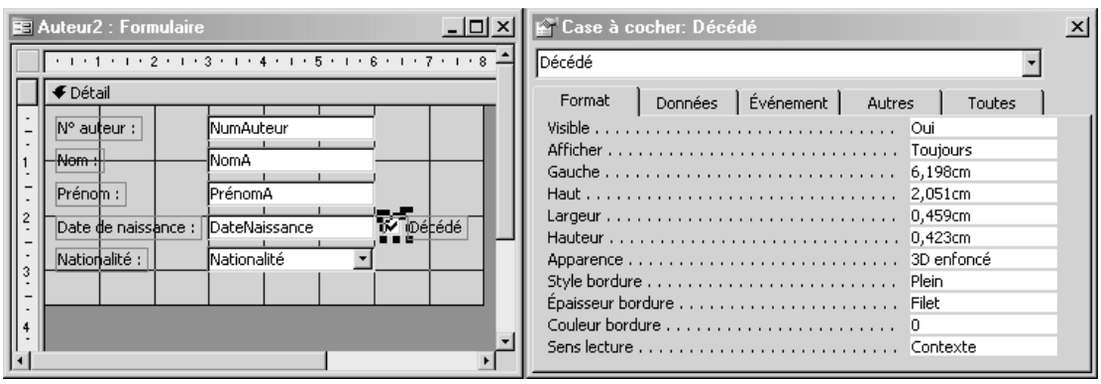
Créons maintenant un formulaire basé sur la table *Auteur2* et faisons glisser les champs à partir de la liste des champs. Il m'a fallu 30 secondes montre en main pour le faire. Voici des copies d'écran du formulaire en mode *Création* et en mode *Affichage* :





Vous constaterez que le champ *Décédé* est bien une case à cocher (cochée par défaut) et que la nationalité est saisie par l'intermédiaire d'une liste modifiable.

Une dernière chose : revenons dans le mode *Création* et cliquons (donc sélectionnons) un des contrôles, par exemple la case à cocher. Voici alors le formulaire et la fenêtre *Propriétés* :

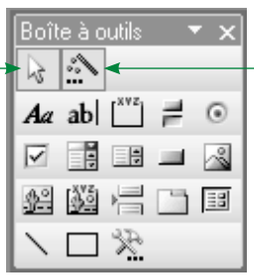


On constate que les contrôles sont bien des objets : la fenêtre *Propriétés* affiche les propriétés de l'objet sélectionné, ici la case à cocher. Vous noterez d'ailleurs que les propriétés sont sensiblement différentes de celles des formulaires.
Enfin, vous remarquerez que le contrôle sélectionné est entouré de carrés noirs. Ce sont les boutons de redimensionnement classiques sous Windows. Vous pouvez donc changer la taille des contrôles comme vous le souhaitez.
Pardon ? Vous voulez une conclusion ? Et bien, euh… je suis pris au dépourvu ! Voyons, voyons… Hum. En conclusion, vous constaterez que, comme promis (séquence 3, paragraphe 4), les propriétés fines des champs définies dans l'onglet *Général* lors de la création de la table sont exploitées pour la génération du formulaire. Ce dernier est ainsi réalisé très simplement. Sans cela, il aurait fallu poser soi-même les contrôles (voir le paragraphe suivant) et configurer leurs propriétés, ce qui serait nettement plus long. Maintenant, imaginez que vous fassiez référence à un champ d'une table dans vingt formulaires. Si les propriétés du champ ne sont pas définies lors de la création de la table, il faudra changer les propriétés des vingt contrôles ! Cela ne serait vraiment, vraiment pas raisonnable.

3F1. La boîte à outils et les assistants des contrôles

Lors de la modification d'un formulaire, vous avez accès à une barre d'outils (appelée *boîte à outils*) qui vous permettra d'ajouter ou de modifier les objets (appelés *contrôles*) placés dans votre formulaire. Si cette barre n'est pas affichée, *Affichage/Boîte à outils* ou bouton dans la barre d'outils principale.
C'est la dernière fenêtre que nous allons étudier. Elle est très importante mais paradoxalement, j'ai relativement peu de choses à dire car elle ne fait que regrouper ce que nous avons vu précédemment.
La boîte à outils est plutôt un *sac à contrôles* : tous les contrôles visuels Windows que vous connaissez (vous les utilisez dans toutes les applications Windows) sont présents. Pour en utiliser un dans votre formulaire, vous cliquez sur le bouton qui le représente puis vous cliquez dans le formulaire… le contrôle est créé. Il ne vous reste plus qu'à modifier ses propriétés dans la fenêtre *Propriétés*.
Pour avoir le nom des différents contrôles, maintenez votre curseur souris sur chaque bouton pour avoir son info-bulle. Vous pouvez également cliquer sur le bouton du contrôle qui vous intéresse puis faire *F1*. Vous aurez une petite aide.

Observons la boîte à outils ci-dessous :



Le premier bouton à gauche (« Sélectionner les objets ») est une flèche. Souvenez-vous… c'est la même flèche que dans la boîte à outils de PowerAMC lorsque nous avons dessiné un MCD. Cliquer sur ce bouton permet de retrouver le curseur souris classique (au contraire des autres boutons qui transforment le curseur en positionneur de contrôle).
En revanche, le deuxième bouton (*Assistants de contrôle*) est votre meilleur ami. Comme son nom l'indique, il active les assistants liés aux contrôles. Ce qui veut dire ? Et bien, mettons-nous en situation : vous voulez mettre un contrôle sur le formulaire. Vous cliquez donc sur le bouton correspondant dans la boîte à outils puis sur le formulaire.
Et là, deux cas possibles :

- si le bouton *assistants contrôle* n'est pas enfoncé, le contrôle est posé sur le formulaire et c'est tout. Il vous appartient de définir toutes les propriétés pour le paramétrer ;
- si le bouton est enfoncé, les assistants sont activés. Dans ce cas, une fois le contrôle posé sur le formulaire, Access lance l'assistant du contrôle qui vous posera des questions très simples afin de paramétrer les différentes propriétés.

Je vous prie de bien vouloir vous synchroniser avec moi vis-à-vis des assistants : nous en avons vu plusieurs (assistants masque de saisie, formulaire…). Voici maintenant les assistants contrôles. Nous étudierons les assistants états dans la séquence 10.
L'assistant a pour objet de vous assister (on est bien avancé !). En pratique, il vous pose des questions très claires et utilise vos réponses pour initialiser les propriétés de l'objet en cours. Techniquement, c'est très simple : il est plus intuitif de répondre à la question *Quelle couleur voulez-vous pour le fond de votre formulaire ?* que d'aller chercher la propriété *Couleur fond* de la section *Détail* du formulaire.

Bien entendu, l'exemple de la couleur du fond n'est pas frappant car c'est une propriété simple. Mais quand il s'agit de paramétrer une liste déroulante, ce que nous ferons tout à l'heure, les questions posées sont plus explicites que les propriétés. Je vous présente maintenant ma position au sujet des assistants. Il y a deux niveaux d'utilisation :

- le non-informaticien qui souhaite réaliser une base Access❶ sera leur esclave : ce sont eux qui définiront les formulaires et les contrôles. La base ne contiendra donc rien qui n'ait été fait par un assistant. Autant dire qu'elle sera très limitée ;
- un informaticien (vous, moi…) va utiliser l'assistant car c'est plus rapide que de définir les propriétés une à une. Cependant, il va le faire en comprenant le fonctionnement de l'assistant. Il pourra donc compléter ou modifier ce que ce dernier aura produit pour l'adapter à ses besoins exacts.

Au final, on retrouve quelque chose déjà vu à l'école primaire : avant d'utiliser une calculatrice, il faut apprendre à calculer à la main. Est-ce une corvée ou un rite initiatique

❶ *Et là, à mon avis, il rêve ! Une base de données fait appel à des connaissances théoriques qu'aucun assistant ne peut remplacer.*

préalable à l'usage de la technologie ? Non ! Il s'agit de comprendre la tâche à réaliser avant d'utiliser un outil qui l'automatise. Si vous sautez cette étape, vous devenez tributaire d'un outil magique que vous ne comprenez pas et cela, c'est dangereux !
Où veux-je en venir avec ce sermon ? C'est une bonne idée d'utiliser les assistants (je le fais dès que possible) mais à condition de ne pas s'arrêter là. Il faut ensuite savoir poursuivre le travail soi-même pour finir de paramétrer le contrôle.

Faisons un exercice pour illustrer cela : la zone de liste déroulante du formulaire précédent a été bâtie d'après les propriétés du champ *Nationalité* définies par un assistant. Nous allons ajouter au formulaire trois zones de liste basées sur la nationalité. Il s'agit donc de faire des clones exacts de celle déjà présente ! Il y a trois techniques possibles.

1re technique
C'est la plus simple. Vous sélectionnez la zone de liste déjà faite. Vous faites un copier/coller… c'est fini ! Comparez les propriétés des deux zones : elles sont identiques. Quand on sait que l'on peut copier un contrôle sur un formulaire et le coller dans un autre… c'est une façon efficace de créer des contrôles identiques lorsque l'on veut accéder au même champ dans des formulaires différents. Mais bon, ce n'est pas ce que l'on veut faire ici.

2e technique
Activez les assistants (cliquez sur le deuxième bouton de la boîte à outils). Cliquez alors sur le bouton de la zone de liste déroulante puis sur le formulaire. Vous observerez deux choses : d'une part, le contrôle est positionné sur le formulaire (

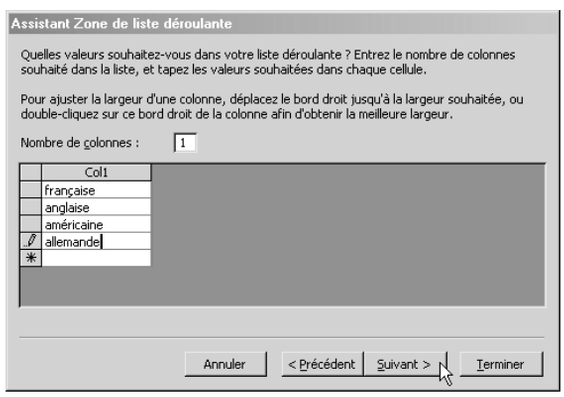
), d'autre part l'assistant est lancé. Notez que le contrôle est pour le moment totalement vierge et n'est lié à aucun champ de la source de données :

1. l'étiquette est *Modifiable10*, à savoir le libellé du type de contrôle❶ et un numéro séquentiel (sans doute un nombre différent chez vous) ;
2. la partie zone de texte de la zone de liste contient *Indépendant* (sous entendu *de tout champ*). Cela signifie que le contrôle n'est pas lié à un champ. Cela pourra changer en fonction du paramétrage que vous allez définir et à condition, bien entendu, que le formulaire soit lié à une source de données. (Car, répétons-le, un contrôle ne peut être lié qu'à un champ de la source de données alimentant le formulaire.)

❶ *Voici une remarque très intéressante mais qui n'aura que les honneurs d'une note de bas de page. Le contrôle qui nous utilisons ici s'appelait historiquement* Zone de liste modifiable. *Il existe deux contrôles de type* zone de liste *: la zone de liste « tout court » et la zone de liste modifiable. La première impose une saisie parmi les valeurs proposées, la seconde autorise la saisie d'une valeur hors-liste (si sa propriété* Limiter à liste *est définie à* Non*). La distinction est faite sur la caractéristique modifiable ou non de la valeur de la liste. Dit autrement, la zone de liste modifiable est une zone de liste à laquelle on ajoute une zone de texte.*
Le problème, c'est que la zone de liste modifiable possède une caractéristique graphique qui a marqué les esprits : le fait que la liste ne présente que sa partie zone de texte, la partie spécifiquement liste ne s'affichant que si l'on clique sur la petite flèche (on a alors l'impression que la liste se déroule). Cette caractéristique a tellement plu au public que le nom impropre zone de liste **déroulante** *s'est imposé. Jusqu'à la version 2000 d'Access, le logiciel ne parlait que de liste* **modifiable**. *Depuis cette version 2003, Microsoft a pris acte du libellé employé par les utilisateurs et, si vous mettez le curseur souris sur la liste dans la boîte à outils, vous verrez* zone de liste **déroulante** *dans l'info-bulle. C'est pourquoi j'ai repris ce nouveau terme dans mon cours. Cela dit, cette mise à jour a été partielle car, dès que vous lancez l'aide sur le contrôle ou sur une de ses propriétés, on vous parlera de zone de liste* modifiable… *Pas d'inquiétude donc, c'est la même chose.*
Et c'est pour cela que, dans ma copie d'écran, ma zone de liste déroulante *s'appelle* Modifiable.

Nous allons maintenant répondre à l'assistant. Il consiste en différentes boîtes de dialogue posant des questions bien senties. Voyons-les une à une.

1. Les valeurs affichées ne viennent pas d'une table. C'est nous qui les entrerons (donc choix 2) ; faites *Suivant*.
2. Cliquer *Suivant* a lancé une nouvelle boîte de dialogue. Nous aurons une colonne et les différentes valeurs à afficher sont *Française*, *Anglaise*, *Américaine*, *Allemande*. Vous les entrerez les unes sous les autres dans la feuille de données :



3. Avant-dernier écran : que faire de la valeur saisie ? Vous pouvez la mémoriser pour un usage ultérieur ou la stocker dans un champ à préciser. Cela n'est pas très clair. Détaillons un peu :
   - *mémoriser la valeur* signifie en fait *ne rien en faire*. La liste ne sera liée à aucun champ de la source de données (en mode *Création*, il y aura toujours *Indépendant* dans la partie zone de texte de la zone de liste). Tant que le formulaire est affiché, vous aurez accès au contenu de la liste (donc à la valeur tapée) par programmation. En effet, la valeur stockée dans un contrôle est une propriété de ce contrôle. Vous pouvez donc la lire ou la modifier par programmation. Voir ci-dessous pour l'intérêt des contrôles indépendants ;
   - la stocker dans un champ revient à ce que l'on faisait en glissant un champ vers le formulaire : on lie le contrôle à un champ de la source de données. Le contrôle affichera donc la valeur du champ associé de l'enregistrement courant de la source. Symétriquement, modifier la valeur du contrôle (en sélectionnant une valeur dans sa partie zone de liste ou en tapant quelque chose dans sa partie zone de texte) modifiera immédiatement et de façon transparente la valeur du champ pour l'enregistrement courant de la source.

   Nous voulons bien entendu lier la valeur saisie au champ *Nationalité*.
4. Le dernier écran vous permet de choisir le texte à afficher dans l'étiquette du contrôle. Nous mettrons « Nationalité : ». Vous observerez la case à cocher vous proposant de l'aide… cochez cette case et étudiez l'aide ! Vérifiez ensuite en activant le formulaire que notre nouvelle zone de liste est identique à la première.

Cliquez sur *Terminer*… c'est fini. Vérifiez ensuite en activant le formulaire que notre nouvelle zone de liste est identique à la première.

3e technique

Désactivez le bouton *Assistants de contrôle*. Recommencez la manœuvre : clic sur le bouton de la zone de liste déroulante puis clic sur le formulaire. C'est fini ! Vous obtenez une zone de liste vierge dont vous devrez définir toutes les propriétés à la main.

Pour observer le chemin qu'il vous reste à faire, comparez les propriétés de cette zone de liste avec celles de la zone paramétrée par l'assistant puis lancez-vous. Cela sera évidemment beaucoup plus simple si vous avez lu l'aide offerte par la boîte précédente (la petite case à cocher).

Exercice

Testez réellement ces trois techniques et passez du temps sur l'assistant pour en étudier les différentes possibilités.
La zone de liste déroulante est le contrôle le plus important et le moins évident à comprendre réellement : un contrôle qui affiche des données mais en enregistre une autre, ce n'est pas si simple.

Les contrôles indépendants

Dans la 2e technique, nous avons parlé des contrôles indépendants. Quel est leur intérêt ? Voici deux situations où l'on peut en avoir besoin :

1. C'est utile si la valeur que vous voulez stocker dans la base est d'un format différent de ce que l'utilisateur a l'habitude de manipuler. Vous utiliserez un contrôle indépendant pour saisir une valeur que vous transformerez par programmation et c'est le résultat que vous stockerez dans la table. Prenons un exemple : vous souhaitez ne stoker que des valeurs monétaires en euros. Cependant, comme beaucoup raisonnent encore en francs, chaque formulaire proposant la saisie d'une somme d'argent comportera deux zones de saisie : une pour la saisie en francs, l'autre pour la saisie en euros. La zone en euros sera liée à la source de données. La zone en francs sera indépendante mais vous la programmerez de telle façon que lorsque l'on tape un montant dedans, ce montant divisé par 6,55957 s'inscrive dans la zone en euros (ce qui stockera automatiquement le montant en euros dans la base).
2. On s'en sert également pour saisir des informations utiles sur le moment mais qui n'ont pas à être stockées. Par exemple, vous voulez saisir un numéro de compte bancaire. À chaque numéro sont associés deux chiffres (la clé). Un calcul mathématique précis sur le numéro de compte doit donner la clé. Si le résultat ne correspond pas, c'est que le numéro est erroné (en pratique, vous avez fait une faute de frappe en le saisissant). Pour autant, la clé ne sert qu'à vérifier la validité du numéro mais n'en fait pas partie. Vous pouvez donc saisir le numéro dans une zone liée au champ *NumCompte* de la source de données et la clé dans une zone indépendante. Par traitement (programmation), vous ferez le calcul sur le numéro de compte et, si le résultat diffère de la clé, vous annulerez la saisie. Ainsi, on utilise la clé pour valider la saisie mais on ne la stocke pas dans la base.

3F2. On continue ?

Non, on arrête ! Nous avons vu beaucoup de choses importantes dans cette séquence. Prenez le temps de bien les assimiler, testez les différentes manipulations. Essayez les assistants des divers contrôles pour deviner leur rôle. Nous les étudierons dans la séquence suivante. Notez que les contrôles les plus simples (intitulé, zone de texte) n'ont pas d'assistant.
Dans la séquence suivante, nous étudierons finement les contrôles et les formulaires. Je vous assure néanmoins qu'en exploitant judicieusement cette séquence, vous pouvez déjà faire des formulaires tout à fait performants.

Synthèse

Nous avons appris beaucoup de choses. Bien entendu, vous devez maintenant savoir à quoi sert un formulaire (affichage et saisie d'enregistrements de tables).

Vous devez également maîtriser :

- les assistants de génération des formulaires ;
- le rôle de la source de données ;
- le principe des propriétés des différents objets d'Access (ici, le formulaire) ;
- les différentes fenêtres accompagnant la création du formulaire.

Séquence 8

Les contrôles graphiques

Nous allons étudier les différents contrôles que l'on peut placer dans un formulaire.

▶ *Capacités attendues*

- **Savoir créer tout type de formulaire (sans programmation)**

▶ *Contenu*

1. Rappels

Dans la séquence précédente, nous avons étudié les contrôles par l'intermédiaire de la boîte à outils et des assistants.
Nous avons vu que tous les objets et en particulier les contrôles possédaient des propriétés accessibles par la fenêtre du même nom. Certaines propriétés sont communes à tous les contrôles, d'autres sont spécifiques à un seul. Voici quelques exemples de ce que l'on peut paramétrer pour chaque contrôle :

- sa taille et sa position dans le formulaire (propriétés *Haut*, *Gauche*, *Largeur* et *Hauteur*). On ne modifie jamais à la main ces propriétés : c'est en déplaçant ou en redimensionnant le contrôle à la souris qu'elles sont mises à jour ;
- la couleur du fond ou de la police (propriétés *Couleur fond*, *Couleur texte*) ;
- le fait qu'il soit visible ou non (propriété *Visible*) ;
- le fait que l'on puisse modifier le contenu de sa zone de saisie (propriété *Verrouillé*).

Si le bouton *Assistants de contrôle* de la boîte à outils est enfoncé, un assistant se déclenchera automatiquement lors du choix de l'un des contrôles ; des questions vous seront posées pour renseigner certaines de ses propriétés. Rien ne vous empêche ensuite de les modifier. Sans l'assistant, toutes les propriétés seront vierges. Ne faites pas le fier et n'hésitez pas à utiliser l'assistant, notamment pour les contrôles complexes (listes déroulantes, sous-formulaires). Il vous sera plus facile de répondre à des questions que de tout définir à la main.
Il est temps de rentrer dans le détail des contrôles. Pour chacun d'eux, nous allons voir :

- comment il fonctionne ;
- comment le paramétrer (c'est-à-dire remplir ses propriétés) ;
- dans quel cas l'utiliser. En effet, chaque contrôle a une forme ou une caractéristique propre qui le rend plus efficace dans la saisie d'un type de données ; par exemple, pour saisir une valeur booléenne (*oui* ou *non*), on peut demander à l'utilisateur de taper au clavier soit *oui*, soit *non*. C'est lamentable, mais techniquement, c'est correct. On peut également utiliser une case à cocher, qui est le composant fait pour cela.

Je vous préviens que je ne compte absolument pas être exhaustif. Il est possible de faire plusieurs pages par contrôle. Mais ce que je dirais alors, vous le trouverez dans l'aide. Je ne vais donc donner que les éléments essentiels.
Vous **devrez** compléter ce cours par la pratique et l'étude de l'aide. Un exemple ? Chaque contrôle (en fait, chaque objet : formulaire, contrôle…) possède de très nombreuses propriétés. Nous verrons celles qui sont essentielles, mais pour toutes les autres, vous devrez consulter l'aide.

Si vous ne le faites pas, vous ne connaîtrez pas tout le potentiel des contrôles. Des choses qui vous sembleront impossibles à faire pourront l'être par des propriétés que vous ne connaissez pas. Comprenez bien que les propriétés caractérisent les objets. De façon naturelle, on peut donc se douter que toutes les caractéristiques intéressantes (utiles) sont représentées par des propriétés.

Je prendrai les contrôles dans l'ordre où ils apparaissent dans la boîte à outils. Notez qu'ils ne sont pas inventés par Access : tous proviennent du système d'exploitation Windows. Ainsi, quelle que soit l'application Windows utilisée (Word ou une application Access que vous avez développée), on retrouve une interface uniforme. Lorsque vous voyez une liste déroulante, vous savez que vous devez choisir une valeur parmi une liste

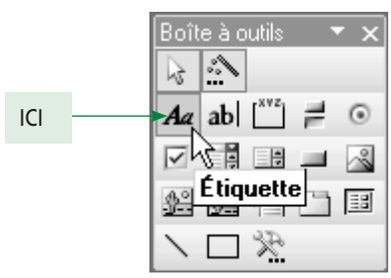
(par exemple, la liste déroulante pour saisir une police sous Word). Lorsque vous voyez une case à cocher, vous savez que vous pouvez ou non activer quelque chose (par exemple l'exposant dans la boîte de dialogue *Police* sous Word).
Notons enfin que si les contrôles sont liés à un champ de la source de données, ils affichent la valeur du champ correspondant de l'enregistrement courant. Si vous modifiez le contenu d'un tel contrôle, c'est la valeur de ce champ pour l'enregistrement courant qui est modifiée.

2. Les contrôles les plus simples

Je vais présenter très rapidement plusieurs contrôles car ils sont très simples d'emploi et l'aide est amplement suffisante si vous avez besoin d'informations supplémentaires.

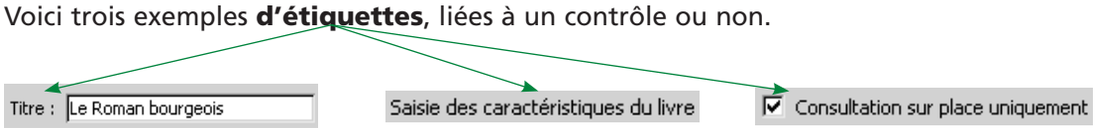
2A. *Étiquette*

Si vous voulez écrire du texte constant dans un formulaire, vous le ferez avec un contrôle *Étiquette* que vous trouverez dans la boîte à outils :



La particularité de ce contrôle est qu'il ne permet aucune saisie (il n'est d'ailleurs associé à aucune zone de saisie). Il ne peut donc pas être lié à un champ de la source de données. Il correspond aux champs pré-imprimés des formulaires papier.
Vous avez déjà rencontré des étiquettes. En effet, tous les contrôles (sauf l'étiquette) possèdent une zone de saisie pour la saisie de l'information et un libellé (du texte) éclairant l'utilisateur sur ce qu'il doit saisir❶. Le libellé est contenu dans un contrôle *Étiquette*.
Lorsque le contrôle est créé automatiquement (en glissant un champ depuis la *Liste des champs* ou par génération automatique du formulaire), c'est la légende du champ ou à défaut son nom qui est utilisé pour remplir l'étiquette.

Voici trois exemples **d'étiquettes**, liées à un contrôle ou non.



2B. *Zone de texte*

Le contrôle *Zone de texte* permet d'afficher ou de saisir des données de tout type (texte, date, chiffre). C'est en définissant les propriétés de la zone de texte (*Format*, *Décimales* et *Masque de saisie*) que vous adapterez la zone au type de contenu.

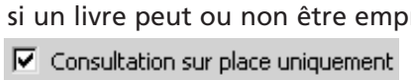
❶ *Nous l'avons vu dans la séquence précédente (paragraphe 3F1).*

Lorsque vous ajoutez un champ dans le formulaire à l'aide de la technique du *glisser-déplacer*, vous obtenez automatiquement une zone de texte pour la majorité des types de données. Les champs *Oui/Non* et *OLE* sont les seules exceptions. Une zone de texte est toujours créée avec un intitulé, que vous pouvez déplacer… ou supprimer.
Dans le paragraphe 2 de la séquence 9, nous verrons une application intéressante de la zone de texte où vous afficherez la valeur des champs, mais aussi des expressions que vous écrirez.

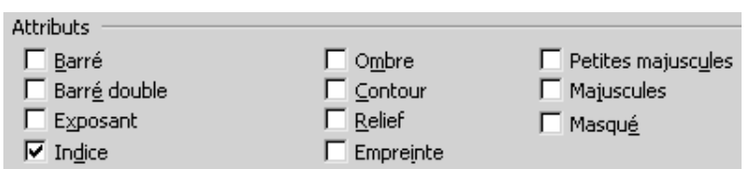
Voici trois exemples de zones de texte, permettant respectivement de saisir une date, du texte et un montant monétaire.

2C. Case à cocher

La *case à cocher* doit être utilisée pour saisir une valeur booléenne autonome.
Comme je le dirai dans le paragraphe sur le bouton bascule, il est préférable d'utiliser une case à cocher pour dire si un livre peut ou non être emprunté, ce qui donne :

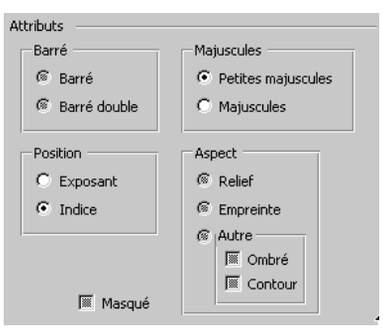


Autre exemple issu de Word (menu *Format/Police…*) :



Nous utilisons des cases à cocher (et non des boutons d'option) car les choix ne sont pas exclusifs : du texte peut être à la fois en indice et en petites majuscules.
Cela dit, certains choix sont exclusifs deux à deux : on ne peut pas être à la fois *Exposant* et *Italique* ou *Barré* et *Barré double*. Suis-je donc en train de renier tous mes principes ?

En appliquant strictement les règles d'emploi des contrôles, voici ce que l'on obtiendrait :



Que penser de cette présentation *new style* ? Ma foi, elle est nulle ! Certes, elle apporte beaucoup d'informations sur le format d'affichage. Le texte peut être :

- soit barré, soit barré double, soit pas barré du tout (les boutons d'option grisés dans un groupe indiquent qu'aucun choix n'est fait) ;
- soit en exposant, soit en indice, soit ni l'un ni l'autre ;
- soit normal, soit en petites majuscules, soit en majuscules ;
- soit en relief, soit en empreinte, soit ombré et/ou en contour.

Cela dit, ces informations sont d'un intérêt très limité et se paient au prix fort :

- le formulaire a triplé de taille ;
- il est beaucoup plus dense que la version originale ;
- j'ai dû utiliser des termes assez artificiels (position, aspect, autre) ou redondants (barré, majuscules) pour nommer les groupes ;
- le choix assez complexe présenté dans le groupe *Aspect* m'a obligé à inclure un groupe dans un autre et à mettre des cases à cocher. La sémantique est correcte : on peut être à la fois *Ombré* et *Contour*. mais pas si on est *Relief* ou *Empreinte*. C'est donc académiquement exact… mais très lourd.

Conclusion ? Et bien, il faut respecter exactement les règles, y compris celle disant que toute règle se transgresse !
Si la fenêtre *Format/Police…* de Word ne suit pas les règles d'ergonomie habituelles, c'est parce que le faire n'apporterait rien, si ce n'est un formulaire illisible.

2D. Bouton de commande

Un *bouton de commande* lance l'exécution d'une macro ou de code VBA (*Visual Basic for Applications*). Cela permet de trier ou rechercher des enregistrements, d'ouvrir un formulaire, mettre à jour une table, imprimer…
Nous l'aborderons dans les séquences de programmation (macro et VBA). Le formulaire présenté dans la séquence 7 (paragraphe 1B) possédait des boutons de commande.

2E. Image

Le contrôle *Image* permet simplement d'ajouter une image sur un formulaire. Je vous déconseille d'afficher des images dans votre application. Cela n'a aucun sens d'un point de vue ergonomie.

2F Cadre d'objet indépendant

Dans un *cadre d'objet indépendant*, vous pouvez insérer un objet indépendant de la source de données. Quand je parle d'objet, ce peut être quelque chose de lourd, comme un document Word ou un classeur Excel. Comme l'objet est indépendant, il ne change en rien lorsque vous passez d'un enregistrement à un autre. Voir l'aide pour plus de détails, cela est hors sujet ici.

2G. Cadre d'objet dépendant

Un *cadre d'objet dépendant* permet d'afficher un objet dépendant de la source de données, donc lié d'une façon ou d'une autre à un champ de la source. Lorsque vous changez d'enregistrement, le contenu du contrôle change.

2H. Contrôle Onglet

Un *contrôle Onglet* permet d'insérer un ensemble d'onglets comportant chacun des contrôles. Vous voulez des exemples de formulaire avec des onglets ? Et bien…

- la fenêtre *Propriétés* (sur chaque onglet, il y a des zones de texte ou des zones de liste déroulante) :

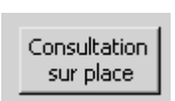
- sous Access, la commande *Outils/Options…* ouvre une boîte de dialogue avec une tripotée d'onglets :

L'ensemble de la boîte de dialogue *Options*, donc les différents onglets, possède 139 contrôles. Il est inimaginable de les présenter tous d'un coup dans le formulaire. L'utilisateur s'y perdrait. Les onglets permettent de structurer les contrôles, donc les informations, ce qui augmente la lisibilité.

2I. Bouton bascule

Les *boutons bascule* permettent de gérer les champs *Oui/Non* en étant enfoncés ou non. Il faut les distinguer des boutons de commande (paragraphe 2D) qui, lorsque l'on clique dessus, exécutent du code VB.
En exemple, je vous propose un bouton bascule indiquant si un livre doit être consulté sur place ou peut être emprunté :

le livre correspondant peut être emprunté



celui-ci doit être consulté sur place

Je vous parle de ce contrôle par politesse. En effet, il ne correspond à rien dans l'ergonomie classique. C'est encore une idée de Microsoft pour faire plaisir aux ignorants qui aiment le concept d'interrupteur. En spécialiste averti, vous utiliserez la case à cocher pour les valeurs booléennes. (Voir ci-dessus le paragraphe 2D.)

2J. Case d'option

Les *cases d'option* (on parle aussi de boutons radio) doivent être utilisés dans un groupe d'options (voir le long paragraphe 3) pour permettre de sélectionner une option et une seule parmi plusieurs.

2K. Sous-formulaire

Le contrôle *Sous-formulaire* permet d'inclure un autre formulaire au formulaire courant. Les deux formulaires doivent être créés avant que l'on puisse les associer.
Nous abordons la notion de sous-formulaire dans la séquence suivante.

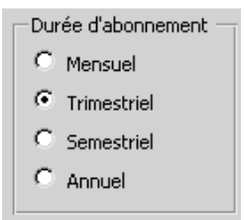
3. Groupe d'options

Nous abordons ici un contrôle complexe. Enfin, très intuitif, mais plus riche que les précédents.

3A. *Présentation*

Un *groupe d'options* contient des cases d'option❶. À l'exécution, on ne peut sélectionner qu'une des options du groupe. On l'utilise donc pour saisir une valeur unique parmi une liste de valeurs prédéfinies (5 maximum). C'est une alternative à la liste déroulante (paragraphe 4).

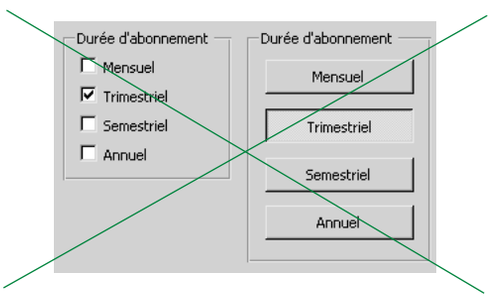
Exemple : pour informatiser un club de sport proposant des abonnements de un, trois, six ou douze mois, vous utiliserez un groupe d'options :



Notez bien que le groupe d'options n'est qu'un conteneur. Son assistant est très utile car il vous demandera quelles valeurs le groupe peut contenir et il les positionnera en les espaçant régulièrement.

J'ai dit que le groupe d'options contenait des boutons d'option. Vous devez en effet vous limiter à ce seul contrôle car son fonctionnement est normalisé en ce sens : une seule option peut être choisie. Hélas, Microsoft a souhaité faire plaisir aux ignorants. L'assistant du groupe d'options vous proposera donc au choix de le remplir avec des boutons d'option, mais aussi des boutons bascule ou des cases à cocher. N'utilisez jamais ces deux derniers types de contrôles car votre interface ne serait plus normalisée. Nous avons vu la case à cocher dans le paragraphe 2C.

Ainsi, ne faites jamais les types de groupe d'options ci-dessous. Le fait que vous les trouviez plus *sympas* n'est surtout pas un argument ! Nous verrons dans la séquence 13 qu'en matière d'ergonomie, l'originalité est une erreur.



Non ! Ces contrôles ne sont pas utilisés dans leur sémantique habituelle.

Ce n'est pas encore fini. Nous allons utiliser l'assistant car ce contrôle possède une caractéristique assez particulière (partagée avec la zone de liste déroulante). L'astuce est la suivante : la valeur affichée (ou sélectionnée) dans le contrôle **n'est pas** la valeur stockée dans la base. En fait, à chaque valeur **affichée** est associée une valeur numérique qui sera **stockée**.

Nous allons expliquer cela sur deux exemples présentant chacun une situation différente.

❶ *Encore une mise à jour du vocabulaire : dans Access 2003, on parle de case d'option. Dans les versions précédentes, et dans l'aide d'Access 2003, vous trouverez bouton d'option. C'est la même chose.*

3B. 1er exemple : groupe d'options Type de livre

Un livre peut être au format poche, relié ou broché. Ces valeurs sont exclusives : un livre broché n'est ni relié ni en poche. Le type le plus fréquent est le poche.
Lorsque je veux saisir le type d'un livre, quel contrôle vais-je utiliser ? Voyons :

- je dois choisir une valeur et une seule parmi plusieurs ;
- j'ai trois valeurs possibles et non cinquante.

Ces deux arguments plaident en faveur d'un groupe d'options (quelle surprise !) sélectionnant la valeur *Poche* par défaut. Le voici :

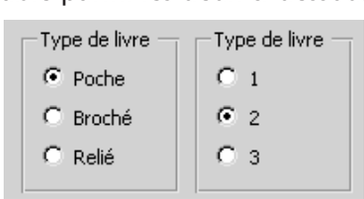
Étudions maintenant le sujet m'amenant à saisir le type d'un livre. C'est une variation de la base *Bibliothèque*. On imagine aisément le MLD (je ne mentionne que ce qui m'intéresse) :

Livre (NumLivre, Titre…, **NumType#**, NumAuteur#)
Type (NumType, LibelléType) *(primaire, étrangère#)*

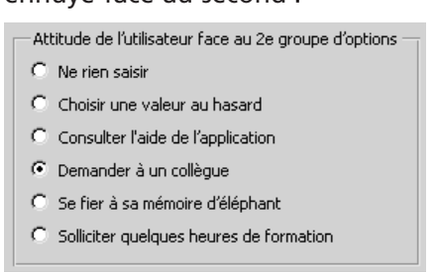
La table *Type* contient les données suivantes :

Table *Type*	
NumType	LibelléType
1	Broché
2	Poche
3	Relié

Je suis en train de créer un formulaire basé sur la table *Livre*. Il me permettra évidemment d'y rentrer des enregistrements.
Lorsque je saisis un livre, je dois entrer toutes ses caractéristiques et notamment son type. Dans la table *Livre*, *NumType* est une clé étrangère. Il faudra donc y stocker la valeur *1*, *2* ou *3*. Bien entendu, si un utilisateur sait visuellement qu'un livre est broché, au format poche ou relié, il ne va pas retenir que c'est le type *1*, *2* ou *3*. D'ailleurs, quel groupe d'options vous semble préférable parmi les deux ci-dessous ?



Difficile de se tromper avec le premier groupe. En revanche, l'utilisateur sera bien ennuyé face au second :



*(La valeur par défaut proposée est pragmatique.
Le troisième choix devrait être le réflexe de l'utilisateur en difficulté.)*

Nous avons un petit souci :

- la valeur stockée dans la base doit être celle de la clé primaire, soit *1*, *2* ou *3*;
- pour le confort de l'utilisateur et la réduction des erreurs de saisie, il faut donner le choix entre des valeurs textuelles (*Poche*, *Broché* ou *Relié*).

Bref, même si c'est le numéro (*NumType*) que l'on stocke, c'est le libellé (*LibelléType*) que l'on veut afficher. Et bien, le groupe d'options permet cela : l'utilisateur a l'impression de choisir le texte *Poche*, *Broché* ou *Relié* alors qu'en fait il saisit respectivement *2*, *1* ou *3*. Comment ce petit miracle se réalise-t-il ? Tout simplement parce que le groupe d'options associe à chacune de ses options une valeur numérique. Pourquoi cela fonctionne-t-il ainsi ? Comme je le dis toujours, parce qu'on en a besoin (la preuve avec notre exemple !). Auriez-vous pu le découvrir seul ? Oui, en observant les propriétés du groupe d'options ou, plus simplement, en utilisant l'assistant.

Nous allons donc utiliser l'assistant pour faire le groupe d'options *Format du livre*.

1. Les étiquettes.

Ce seront les valeurs affichées :

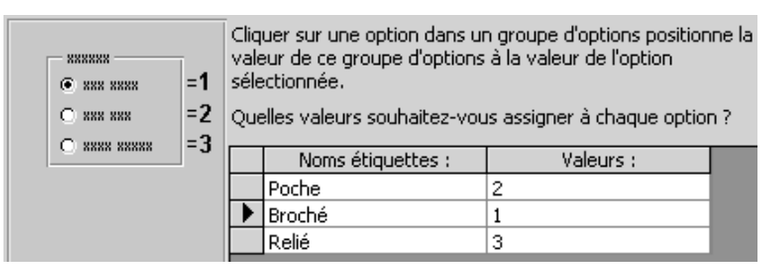
	Noms d'étiquette :
	Poche
	Broché
	Relié
*	

L'ordre des valeurs est important. Ici, je place la valeur la plus fréquente en tête puis j'utilise l'ordre alphabétique. Cet ordre n'est pas celui de la table *Format* mais cela n'a aucune importance : dans une table, les données ne sont jamais triées (elles sont stockées dans leur ordre de saisie). En revanche, lorsque l'on affiche des données (requête ou autre), on les présente dans l'ordre le plus efficace.

2. Je mets la valeur par défaut *Poche* car la majorité de mes livres sont de ce format.

3. La boîte de dialogue suivante est le nœud de l'affaire : on indique la valeur **numérique** associée à chaque option. C'est elle qui sera la valeur du contrôle et sera stockée dans la base. Je dois donc utiliser les valeurs de la clé primaire *NumFormat*. Les valeurs proposées par défaut *1*, *2*, *3…* me conviennent mais pas leur ordre. Je change donc en 2 (*Poche*), 1 (*Broché*) et 3 (*Relié*).

Vous noterez que la boîte, dont je copie ci-dessous une partie, est explicite et ma foi, fort bien faite.



4. On vous demande ensuite si le contrôle est indépendant ou lié à un champ de la source de données. Dans mon cas, il sera lié au champ *NumFormat* de la source de données basée sur *Livre*. (Je vous rappelle que je suis en train de saisir un livre.)

5. Ensuite, des petits réglages visuels; je vous conseille les cases d'option et le style échelonné.

6. Enfin, vous devez saisir la légende, c'est-à-dire le libellé du contrôle. Ici, nous entrerons *Format du livre*.

Notez la case pour l'affichage de l'aide. Activez-la et lisez l'aide !

Comme d'habitude, je vous conseille d'aller voir les propriétés du groupe d'options pour observer comment ce que vous avez paramétré par l'assistant a été stocké.
J'insiste, faites-le vraiment. Il y a une grande différence entre comprendre ce cours et assimiler son contenu pour pouvoir refaire ce qui y est présenté. Pour assimiler, il faut lire, relire, tester, retester, essayer, bidouiller…

3C. 2e exemple : groupe d'options Durée d'abonnement

Nous souhaitons saisir la durée d'abonnement à un club de sport (l'abonnement peut être mensuel, trimestriel, semestriel ou annuel).
Ici, le problème est différent car la table *Abonnement* du MLD sera :

Abonnement (NumAbonnement, **DuréeMois**, NumClient#…) primaire, étrangère#

C'est le champ *DuréeMois* que nous voulons saisir. Il n'y a donc plus de notion de clé étrangère. La valeur saisie sera *1*, *3*, *6* ou *12* selon que l'abonnement est mensuel, trimestriel, semestriel ou annuel. Nous pourrions utiliser une zone de texte pour saisir le chiffre de la durée en mois. Mais, bien entendu, ce n'est pas acceptable pour deux raisons :

- c'est assez fastidieux ;
- surtout, l'utilisateur ne saura pas quelles sont les valeurs autorisées (1, 3, 6 et 12) et celles interdites (2, 4, 5…).

Pour avoir un formulaire soigné, nous utiliserons un groupe d'options. La différence avec l'exemple précédent, c'est que les valeurs associées aux différents choix ne seront plus la clé étrangère mais la durée en mois de l'abonnement. La 3e boite de dialogue de l'assistant sera remplie ainsi :

	Noms étiquettes :	Valeurs :
	Mensuel	1
	Trimestriel	3
	Semestriel	6
▶	Annuel	12

Voilà ! C'est sympa, non ?
À force de relire ce cours, de nouvelles idées m'assaillent : la saisie de la durée est faite en choisissant une valeur textuelle. Je n'y avais pas pensé avant, mais on pourrait tout aussi bien utiliser des nombres. La lecture du contrôle y gagne en rapidité. Cela donne :

3C1. Conclusion sur le groupe d'options

Les valeurs proposées dans le groupe doivent être dans un ordre logique : du plus petit au plus grand, par ordre alphabétique ou par ordre d'usage :

- les durées d'abonnement étaient présentées de la plus courte à la plus longue ;
- pour le format du livre, j'ai mis la valeur la plus fréquente (*Poche*) en premier. Les deux autres sont triées par ordre alphabétique.

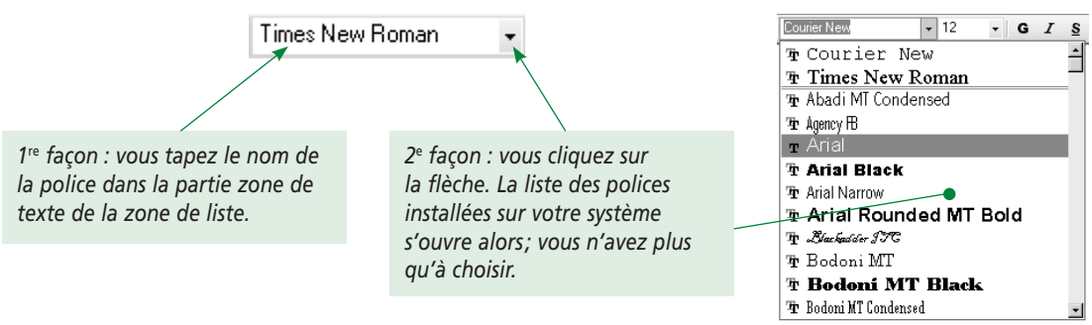
Reprenons le groupe *Format du livre*. Vous remarquerez que les valeurs de chaque case d'option (pour l'affichage et le stockage) sont fournies *en dur* et non issues d'une requête basée sur la table *Format*. Si je voulais gérer un nouveau format, par exemple

Livre électronique, vous ne pourriez vous contenter de le rajouter dans *Format*. Il faudrait également ajouter une nouvelle case d'option dans le groupe (ou, plus simplement, supprimer ce dernier et le recréer). Le groupe avec quatre options serait évidemment plus grand, ce qui risquerait d'obliger à repenser l'organisation du formulaire.
La mise à jour d'un groupe d'options représente un certain travail. Il ne faut donc utiliser ce contrôle que pour des choix stables. L'exemple typique est le titre de civilité d'une personne (M., M^me^ ou M^lle^). Si les choix évoluent (par exemple, le choix d'un auteur pour un livre), on emploiera une zone de liste déroulante (voir le paragraphe 4).
De plus, le groupe est un contrôle volumineux et ne possède aucune aide à la sélection de l'option. Il ne faut donc pas l'employer pour choisir parmi 20 valeurs, mêmes stables : il occuperait tout l'écran et l'utilisateur passerait dix minutes à chercher la valeur qu'il souhaite. Dans ce cas, vous utiliserez la zone de liste déroulante ou la zone de liste *tout court*.
Enfin, les valeurs associées aux options (et donc la valeur du groupe lui-même) ne peuvent être que des entiers. Pour stocker tout autre type de donnée, on utilisera la z_ _ _ d_ l_ _ _ _ d_ _ _ _ _ _ _ _ _. (Je vous laisse le soin de deviner à quel contrôle je pense…)
Vous venez de prononcer l'expression magique *zone de liste déroulante*. Comme je perçois votre intérêt pour ce bel objet, j'en fais le thème du paragraphe suivant.

4. Zone de liste déroulante

4A. Présentation

Une *zone de liste déroulante* est une zone de texte munie d'une flèche. Prenons l'exemple de la liste déroulante permettant de choisir une police sous *Word*. Elle peut être renseignée de deux façons :



On va retrouver un fonctionnement déjà vu avec le groupe d'options : la valeur affichée n'est pas la valeur stockée par le contrôle. Concrètement, dans la liste des polices, je pense avoir choisi la police *Times New Roman* alors qu'en fait le système a retenu que j'avais choisi la police numéro 8. Nous allons étudier trois exemples illustrant différentes situations.

4B. 1er exemple : la nationalité d'un écrivain

Retournez voir le paragraphe 3F de la séquence précédente : nous avions utilisé l'assistant dès la création de la table. Il avait permis d'indiquer que les valeurs du champ *Nationalité* pouvaient faire partie d'un ensemble de valeurs. Je dis bien *pouvaient* et non *devaient* puisque l'on avait autorisé des saisies extérieures à celles proposées.

Nous n'avions alors qu'une seule colonne dans la zone de liste. Cela signifie que le MLD était Auteur(NumAuteur..., Nationalité...), sous-entendu que *Nationalité* était une chaîne de caractères et non une valeur numérique clé étrangère correspondant à une table *Nationalité*.
Je ne reviens pas sur ce cas que nous avons déjà abordé, d'autant qu'il est peu fréquent. Notons cependant que, même en énumérant vous-même les valeurs, vous pouvez avoir plusieurs colonnes comme nous allons le voir ci-dessous.

4C. 2e exemple : l'éditeur d'un livre

Nous allons maintenant travailler sur l'éditeur du livre selon le MLD suivant :

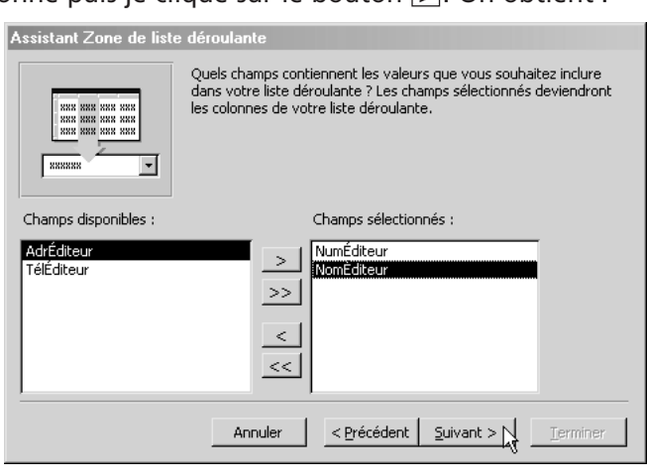
Livre (NumLivre, Titre…, NumAuteur#, NumÉditeur#)

Éditeur (NumÉditeur, NomÉditeur, AdrÉditeur, TélÉditeur) primaire, étrangère#

On retrouve la problématique du groupe d'options : pour saisir un livre, on veut choisir le nom de l'éditeur alors que c'est son numéro qui sera stocké dans la clé étrangère *NumÉditeur*. Il faut donc avoir des valeurs à afficher (les noms) et d'autres à stocker (les numéros).
Comme il y a potentiellement de nombreux éditeurs et que la liste n'est jamais close, pas question d'utiliser un groupe d'options. Nous allons donc nous servir de la zone de liste déroulante que nous créerons facilement avec l'assistant. Bien entendu, nous aurions pu (voir dû) utiliser l'assistant zone de liste dès la création du champ dans la table. Mais bon… On peut aussi (heureusement !) l'utiliser avec les contrôles lors de la création du formulaire.
Vous connaissez la méthode : en vous assurant que le bouton *Assistants contrôle* est enfoncé, vous cliquez sur le bouton *Zone de liste déroulante* puis vous cliquez dans le formulaire.

Nous répondrons ainsi à l'assistant :

1. Les valeurs seront issues d'une table (la table *Éditeur*) donc choix 1.
2. Les valeurs viendront de la table *Éditeur*.
3. J'ai alors accès à tous les champs de la table *Éditeur* dans la zone de liste gauche. Ceux qui m'intéressent sont *NumÉditeur* (valeur qui sera stockée par la liste) et *NomÉditeur* (valeur qui sera affichée). Pour les mettre dans la zone de liste droite, je les sélectionne puis je clique sur le bouton [>]. On obtient :



4. La boîte suivante me permet de trier le contenu de la liste. Je trierai sur *NomÉditeur*.

5. La boîte qui suit est très importante. Elle est liée au choix des champs à afficher.

Access vous recommande de cacher la clé primaire *NumÉditeur*. Attention, *cacher* ne veut pas dire *supprimer* : comme vous avez sélectionné les deux champs *NumÉditeur* et *NomÉditeur* dans l'avant-dernière boîte de dialogue, la liste possédera deux colonnes, l'une contenant *NumÉditeur*, l'autre *NomÉditeur*. Simplement, si vous cachez la clé primaire, la colonne qui la contient aura une largeur nulle. On ne la verra donc pas❶.

Voyons maintenant pourquoi la question se pose de cacher ou pas cette clé :

- la clé primaire doit être présente dans la liste car c'est sa valeur qui sera affectée au contrôle lié à la clé étrangère ;
- cela dit, la valeur de la clé n'est pas intelligible pour l'utilisateur : si Flammarion est un éditeur intelligible, l'éditeur numéro 2 n'est pas du tout parlant. Il ne sert donc à rien d'afficher la clé primaire puisque sa valeur n'apporte aucune information à l'utilisateur.

Pour résumer, on aura besoin d'au moins deux champs dans toute liste permettant de saisir une clé étrangère :

- la clé primaire associée, présente mais masquée car elle contient une information technique mais sans objet pour l'utilisateur ;
- au moins un champ parlant (un nom, un libellé…) pour identifier la saisie aux yeux de l'utilisateur. On peut mettre plusieurs champs en cas de risque d'homonymie. Par exemple, pour saisir un client, on mettra au moins les champs *NomCl* et *PrénomCl* (voire en plus *TéléphoneCl*).

Ne vous laissez donc pas attendrir par cette boîte de dialogue : si Access propose de montrer la clé primaire (même en le déconseillant), ce n'est pas parce que cela peut être utile mais uniquement pour ne pas traumatiser un non-informaticien qui ne comprend pas vraiment comment fonctionne une zone de liste.

6. Le contrôle n'est pas indépendant : il faut le lier au champ *NumÉditeur* de la source de données (basée sur la table *Livre*).

8. Pour l'étiquette, on tapera « Éditeur : ». Comme d'habitude, cochez la case d'aide et lisez-la !

9. C'est fini.

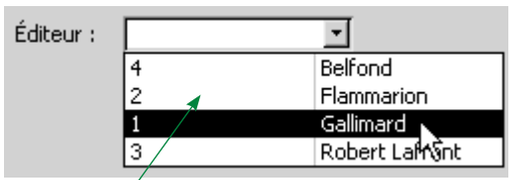
Nous retiendrons que :

- le premier champ énuméré parmi ceux affichés lorsque la liste est déroulée est celui qui s'affiche dans la zone de saisie ;
- pour ne pas faire apparaître un champ, la taille de sa colonne doit être nulle ; en pratique, vous cacherez toujours les clés primaires qui sont des informations d'informaticien et pas d'utilisateur.

❶ *Vous avez exactement le même principe sous Excel avec la commande* Format/Colonne/Masquer *qui permet d'affecter une largeur nulle, donc de cacher, les colonnes contenant des calculs intermédiaires qui ne sont pas utiles à l'utilisateur. (Par symétrie, vous avez aussi* Format/Ligne/Masquer *sous Excel… mais pas sur Access : une ligne étant un enregistrement de la source, si vous n'en voulez pas, vous utiliserez une source requête dont vous travaillerez la clause* where.*)*

4D. Manipulons les propriétés de la liste

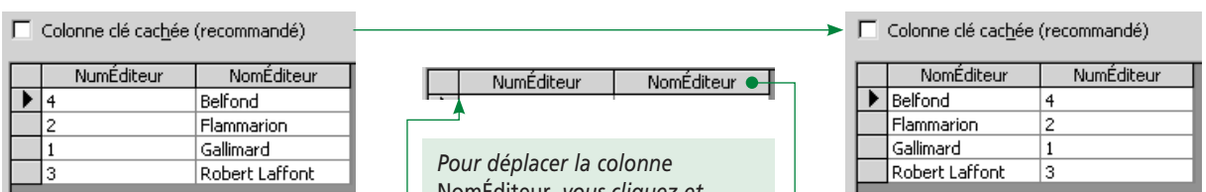
Voyons ce qui se passe dans le formulaire si, malgré tous mes conseils, vous affichez la colonne clé primaire. La zone de liste déroulante aura deux défauts majeurs.



2e défaut : *la valeur affichée par la liste lorsqu'elle est fermée est le numéro de l'éditeur et pas son nom.*

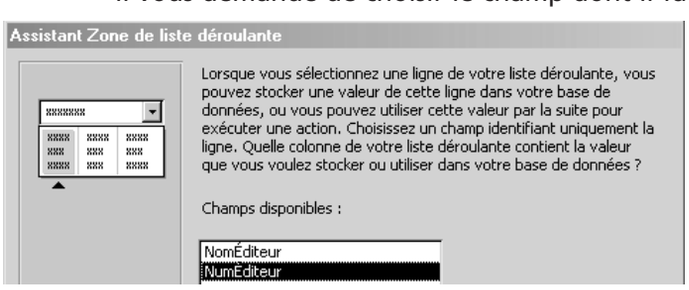
1er défaut *: lorsque l'on déroule la liste, le numéro apparaît. Or, on sait depuis longtemps que le numéro identifiant (la clé primaire) est une valeur technique que l'utilisateur n'a pas à connaître.*

Résumons le problème : nous avons besoin du numéro d'éditeur car c'est cette valeur que nous stockerons, mais nous ne voulons pas la voir affichée, ni dans la zone de saisie ni dans la partie déroulante.
Comment régler le problème ? La liste peut afficher autant de colonnes que nécessaire lorsqu'elle est déroulée. En revanche, elle n'en affichera qu'une seule lorsqu'elle est fermée. Ainsi, même si plusieurs champs sont présents dans la liste, un seul sera affiché. Mais comment le choisir ? Nulle part dans l'assistant la question n'a été posée. En fait, Access se contente d'utiliser la première colonne de la zone de liste déroulante.
Du coup, dans l'assistant proposant de cacher la clé primaire, comme on voit la « tête » de la liste, il suffit de permuter les deux colonnes. (Si vous n'êtes plus dans l'assistant, relancez-le. Sinon, vous pouvez cliquer sur *Précédent* jusqu'à retomber sur cette boîte.)



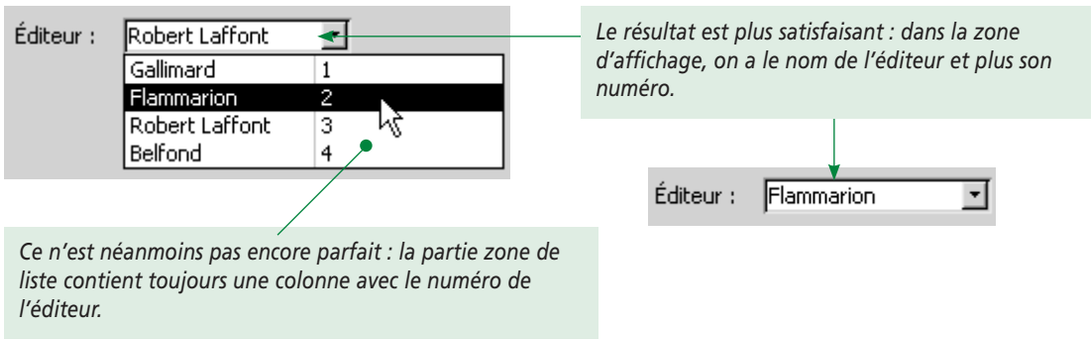
Pour déplacer la colonne NomÉditeur, *vous cliquez et maintenez son en-tête* **ici** *puis vous déplacez le curseur souris...***là.**

Si vous faites cela, Access détecte que la première colonne n'est plus la clé primaire, donc il vous demande de choisir le champ dont il faudra stocker la valeur :



Cette boîte n'apparaissait pas précédemment car Access piochait d'autorité le premier champ (la clé primaire) comme valeur de la liste. Comme l'ordre a changé, il vous laisse la responsabilité du choix.
Nous prenons toujours NumÉditeur.

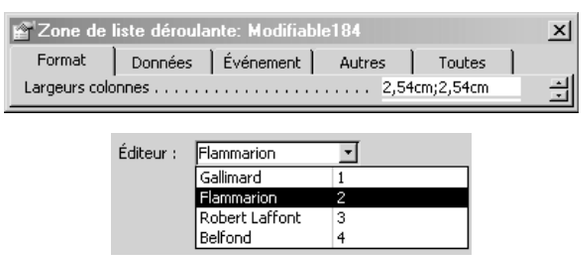
Voici des copies écran de la zone de liste déroulante obtenue :



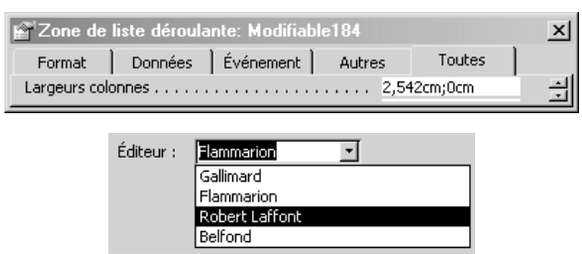
Comment régler ce dernier problème ? L'assistant ne demande jamais quelle colonne doit être affichée. C'est une grosse astuce : pour ne pas voir une colonne, il faut lui affecter la largeur *0 cm* (comme sous Excel). Dans notre cas, si vous cochez la case *Colonne clé cachée*, Access attribue la largeur *0 cm* à cette colonne.

Notez que cela peut aussi se faire avec l'assistant ou avec la propriété *Largeurs colonnes* une fois la zone de liste achevée :

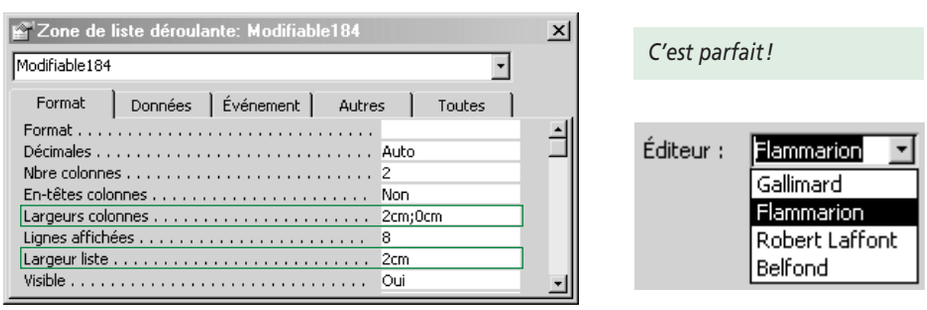
Les deux colonnes font environ 2,5 cm



La 2e colonne contenant le numéro a une taille nulle



Quitte à tripatouiller les propriétés, vous notez que les deux parties listes ont la même largeur ; à droite, comme il n'y a plus qu'une colonne, cela fait un peu grand. Pourquoi la largeur ne s'est pas actualisée seule ? Parce que c'est une propriété autonome, *Largeur liste*. Du coup, je réduis la première colonne et la liste à deux centimètres et j'obtiens ceci :



C'est parfait, certes, mais bon, nous venons de paramétrer à la main la liste pour obtenir exactement le résultat que proposait l'assistant. En pratique, nous venons de cacher la colonne clé caché à la main.

Exercice

Il est indispensable de faire une vraie pause dans le cours en prenant le temps de tester les différents paramétrages de l'assistant, d'étudier l'aide et de manipuler toutes les propriétés disponibles. Faites quelques manipulations, consultez l'aide, bref, expérimentez !

4E. 3e exemple : l'auteur d'un livre

La partie du MLD qui m'intéresse est la suivante :

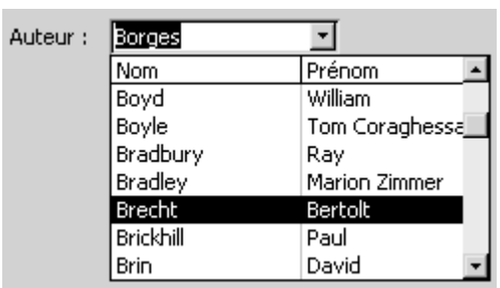
Livre (NumLivre, Titre…, NumAuteur#)

Auteur (NumAuteur, Nom, Prénom) primaire, étrangère#

Dans la table *Livre*, je dois stocker un numéro d'auteur alors que pour l'ergonomie, je voudrais saisir son nom. C'est pour l'instant exactement le même problème que dans le paragraphe précédent.

Pour agrémenter la saisie et réduire les problèmes d'homonymie, je voudrais afficher à la fois le nom et le prénom dans la zone de liste. Pas de problème, il suffit de prendre les trois champs *NumAuteur*, *NomA* et *PrénomA* (clé primaire cachée).

Cela donne le résultat suivant (j'ai mis *Oui* à la propriété *En-têtes colonnes*❶ de la zone de liste) :



Bon, ce n'est pas mal. Mais j'aimerais bien avoir le nom et le prénom ensemble et pas sur deux colonnes. Bref, j'aimerais avoir cette liste :

Notez bien la différence : il n'y a plus qu'une colonne au lieu de deux, mais elle contient le nom et le prénom de l'auteur. Idem dans la zone de saisie de la liste.

❶ *Ce qui ne doit pas vous poser de problème si vous avez fait l'exercice qui précède.*

Comment ai-je fait cela ? Il y a une petite astuce élégante : à partir de l'assistant (2e fenêtre), je dirai que les valeurs sont issues d'une requête que j'aurai pris soin d'écrire et d'enregistrer avant :

```
select NomA & " " & PrénomA AS [Auteurs disponibles], NumAuteur
from Auteur
order by 1 ;
```

(Testez cette requête : elle renvoie bien la concaténation du nom et du prénom.)

4F. Quelques propriétés de la liste déroulante

Les propriétés de ce contrôle sont nombreuses et permettent une gestion très fine. J'ai pioché les plus importantes dans les différents onglets (toutes sont également accessibles par l'onglet *Toutes*). On retiendra notamment :

- *Limiter à liste*
  Si *Oui*, interdit de saisir une valeur qui n'est pas dans la liste. Cela est important si vous saisissez une valeur représentant une clé étrangère. Il faut alors impérativement que la saisie soit faite parmi les valeurs de la clé primaire pour éviter le problème d'intégrité référentielle.
- *Contenu*
  Les valeurs dans la liste peuvent être fournies *in extenso* (cas de la civilité : *M.*, *Mme* ou *Mlle*) ou être le résultat d'une requête à fournir.
- *Nbre colonnes, largeurs colonnes, colonne liée*
  Dans le cas où la liste est alimentée par une requête, vous pouvez choisir quel champ doit être pris en compte pour renseigner le contrôle : c'est la colonne liée. Si la liste est alimentée par une table, c'est la clé primaire qui sera automatiquement retenue avec l'assistant.
  Pour masquer une colonne (généralement la clé primaire), on lui affecte une largeur nulle.
- *En-têtes colonnes*
  Utilise le nom des champs de la table ou de la requête comme en-tête.
- *Lignes affichées* (*8* par défaut)
  Nombre de lignes constituant la partie déroulante.
- *Auto étendre* (*oui* par défaut)
  Le principe est le suivant : quand vous commencez une saisie dans la liste, cette dernière compare ce que vous tapez avec les valeurs qu'elle a en stock. Dès qu'elle trouve un élément qui correspond, elle l'affiche. Par exemple, dans la liste des polices vue au paragraphe 4A, dès que vous tapez *ar*, *Arial* va s'afficher. De même, dès que vous tapez *Be*, *Beeskness ITC* va s'afficher. C'est un gain de temps énorme lors de la saisie : vous n'avez pas à entrer la valeur complète mais juste assez de caractères pour identifier une valeur unique. Vous activerez systématiquement cette propriété.

Comme d'habitude, utilisez l'aide pour plus d'explications !

4G. Comparaison liste déroulante contre groupe d'options

Aurions-nous pu utiliser un groupe d'options pour saisir l'auteur ? On retrouve bien les deux valeurs, celle qui est affichée et celle qui est stockée. Mais cela n'est pas suffisant. Deux arguments empêchent l'usage du groupe d'options :

- ma liste déroulante gère plus de 700 auteurs. Il n'est absolument pas envisageable de faire un groupe d'options avec 700 valeurs ! Ce serait bien entendu trop gros et très peu pratique car on ne dispose pas de la fonction *auto étendre* ;
- surtout, les données affichées dans le groupe d'options sont figées car définies lors de sa création. Au contraire, la zone de liste propose d'aller chercher ses valeurs dans une requête. Si l'on ajoute des auteurs, la requête renvoyant leur nom (et donc la zone de liste basée dessus) les prendra en compte.

La différence essentielle entre ces deux contrôles (aspect visuel mis à part) est donc la faculté pour la zone de liste de prendre dynamiquement ses valeurs dans la base alors que celles du groupe d'options sont définies en dur.

5. Zone de liste

Partez d'une zone de liste déroulante. Enlevez sa partie zone de texte (là où l'on pouvait saisir des valeurs au clavier). Il ne reste que la liste. Cela donne une zone de liste. L'assistant est d'ailleurs similaire. Voici la saisie de l'auteur avec une zone de liste (je me suis basé sur la même requête que précédemment) :

J'ai conservé un en-tête de colonne. C'est discutable. Voire, avec un peu de recul, ce n'est pas super joli. Enfin, c'est une simple propriété à basculer.

On peut toujours mettre plusieurs colonnes ; appuyer sur une lettre vous amène sur la première valeur de la liste commençant par ce caractère (il faut ensuite utiliser l'ascenseur pour se positionner sur la valeur cherchée).
Au final, ce qui distingue la liste de la liste déroulante tient en deux points :

- comme la liste ne possède pas de zone de saisie, le choix est limité aux valeurs proposées ;
- la liste est par définition toujours déroulée. Elle occupe donc beaucoup plus de place mais affiche spontanément ses valeurs, ce qui permet une saisie plus rapide.

Exercice

Créez la zone de liste présentée ci-dessus.

Séquence 9

Les formulaires accèdent aux données

Après avoir vu les différents contrôles, nous allons les lier aux données de la base.

▶ *Capacités attendues*

- **Savoir créer tout type de formulaire (sans programmation)**

▶ *Contenu*

1. Contrôle dépendant ou indépendant ?

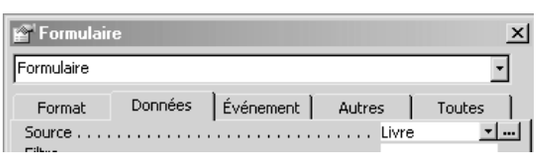
Nous ne l'avions pas formalisé car c'est l'assistant qui s'en chargeait… dans quelles propriétés indiquons-nous pour un contrôle **dépendant** :

- à quelle source de données il est lié ;
- à quel champ de la source il est associé ?

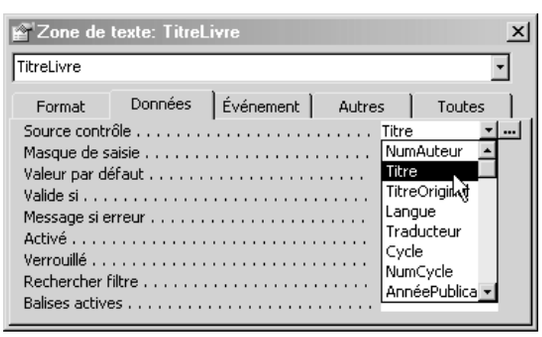
Cherchez sur un contrôle quelconque avant de lire la réponse qui suit.

En fait, il y a un piège dans ma question : rappelez-vous que c'est le formulaire qui est lié à la source de données (par sa propriété *Source* que l'on trouve sous l'onglet *Données* de la fenêtre *Propriétés*). Imaginez un tuyau partant du formulaire branché sur cette source. Si un contrôle du formulaire est lié à une source, ce ne peut être qu'à celle reliée au formulaire.
Dans un formulaire indépendant (lié à aucune source de données), tous les contrôles seront indépendants. Si le formulaire est lié à une source, chaque contrôle du formulaire peut être lié à un champ de cette source ou être indépendant.
La question est alors singulièrement réduite : soit un contrôle est indépendant, soit il est lié à un champ de la source de données alimentant le formulaire.
Pour associer un champ de la source de données au contrôle, on renseigne sa propriété *Source contrôle* (sous-entendu : quelle source alimente le contrôle ? Attention, la source dont on parle ici est un champ, pas une requête ou une table). Illustrons cela sur :

La source d'un formulaire



La source d'une zone de texte



Ces deux copies écran montrent que :

- le formulaire est alimenté par la table *Livre* (enfin, je sais que c'est une table, mais ce pourrait être indifféremment une requête) ;
- ma zone de texte est liée au champ *Titre* de la source de données. Vous observerez que la propriété *Source contrôle* est renseignée par une zone de liste déroulante. Je l'ai déroulée dans la copie écran pour que vous puissiez vérifier que les champs disponibles sont ceux de la source de données du formulaire (soit la table *Livre*).

Pour avoir un contrôle indépendant, il suffit de ne pas mettre de champ de la source de données dans la propriété *Source contrôle*.
Pour autant, faut-il ne rien mettre dans cette propriété ? Cela dépend, car elle sert à indiquer quelle valeur est affichée. Au final, nous avons trois cas. Dans *Source contrôle*, vous pouvez :

- mettre un champ de la source de données. Le contrôle est alors par défaut accessible en lecture et écriture. Bref, il affiche la valeur d'un champ et permet de la modifier ;
- ne rien mettre. Dans ce cas, le contrôle n'affiche rien. Cela ne vous empêche pas de saisir une valeur dans le contrôle ;

- mettre une expression (en tout cas, autre chose qu'un champ). Le contrôle est alors en lecture seule : il affiche la valeur d'une expression. Vous ne pouvez pas modifier cette valeur (pour la même raison que, 2+3 valant 5 et pas autre chose, vous ne pouvez modifier ce chiffre !).

Le premier cas a été étudié tout au long des deux séquences précédentes. Les deux autres seront abordés dans le paragraphe suivant.

2. Zones de texte et affichage d'expressions

2A. Rappel

La zone de texte nous permet actuellement de saisir ou d'afficher des valeurs provenant d'un champ de la source de données.
Comme tout contrôle peut être indépendant, vous pouvez également saisir ou afficher dans la zone de texte des valeurs sans rapport avec la source de données. À quoi cela peut-il servir ?

- saisir une valeur indépendante est utile si cette valeur est modifiée par programmation pour donner une valeur qui, elle, sera stockée (par exemple, vous saisissez une valeur en francs et c'est sa contrepartie en euros qui est calculée et stockée) ;
- afficher une valeur indépendante est utile pour l'ergonomie ou le contrôle de la saisie : pour saisir une facture, vous tapez les montants HT et TTC et le formulaire affiche le montant de TVA. Cette valeur calculée (TVA = TTC – HT) n'est pas stockée dans la base, mais en la comparant avec la TVA de la facture, vous pouvez vérifier que vous n'avez pas fait de faute de frappe. La valeur de TVA affichée n'est pas modifiable (c'est logique puisqu'elle est issue d'un calcul).

Nous l'avons vu dans le paragraphe précédent : si l'on saisit une valeur indépendante, c'est que la propriété *Source contrôle* est vierge. Il n'y a donc rien de plus à dire ! Si l'on affiche une valeur indépendante (en lecture seule), il faut mettre l'expression calculant la valeur dans la propriété. Nous allons le voir maintenant.
Même si tous les contrôles peuvent être indépendants, le cas le plus fréquent reste la zone de texte puisqu'elle permet d'afficher facilement tout type de données.

2B. Affichage d'expressions

Notre objectif est le suivant : utiliser des zones de texte pour afficher *quelque chose*, à savoir le résultat d'une expression.
J'utilise volontairement le terme *expression* car il s'agit exactement des expressions utilisées dans les formules Excel. Une expression est d'un type quelconque (texte, numérique) et se calcule (s'évalue) pour donner une valeur. C'est cette valeur qui sera affichée.
L'expression sera écrite dans la propriété *Source contrôle* de la zone de texte. Pour indiquer que l'on mentionne une expression et pas un champ de la source de données, toute expression commence par le caractère « = » (comme une formule Excel).

2C. Contenu d'une expression

2C1. Principe

Une expression peut utiliser :

- du texte (une chaîne de caractères) qui sera, comme d'habitude, mis entre guillemets ;
- des valeurs numériques ;
- les valeurs d'autres contrôles du formulaire (voire venant d'autres formulaires) ;
- les valeurs des champs de la source de données.

Pour uniformiser tout cela, on utilisera :

- l'opérateur « & » pour concaténer les valeurs texte ou numérique ;
- les opérateurs arithmétiques habituels pour faire des calculs.

2C2. Un exemple simple

Un exemple tout simple : je veux afficher le texte *Bonjour à tous*. Les expressions suivantes sont valides (la première est bien entendu préférable !) :

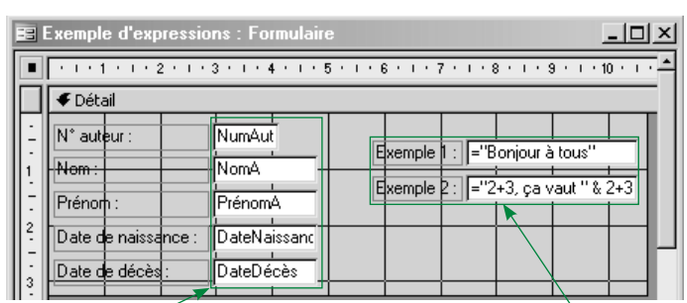
- ="Bonjour à tous" ;
- ="Bonjour" & " " & "à" & " " & "tous".

Je vous rappelle une dernière fois que ce texte doit s'écrire dans la propriété *Source contrôle*. On retrouve le contenu de cette propriété dans la zone de texte en mode *Création* du formulaire. C'est bien pratique pour savoir ce que l'on affichera !

Une dernière remarque : si vous trouvez la zone de saisie des propriétés un peu courte, appuyez sur *Maj+F2* lors de la saisie et une fenêtre plus confortable s'ouvrira.

Voici une illustration du fonctionnement.

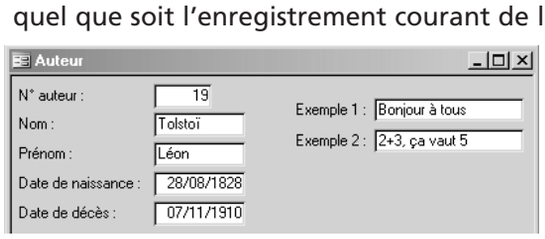
Mode *Création* :

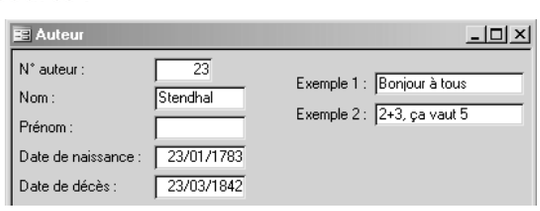


On voit que ces contrôles sont liés (pas de caractère « = »)

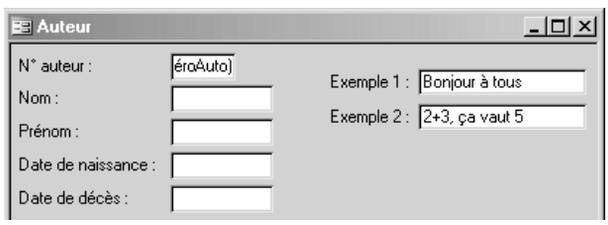
Voici deux zones de texte indépendantes

En mode *Formulaire*, le contenu des zones de texte indépendantes demeure constant quel que soit l'enregistrement courant de la source :





Même lors de la saisie d'un nouvel enregistrement, nous retrouvons nos messages :



Bien. Vous noterez que les expressions utilisées sont des constantes (c'est pourquoi elles affichent toujours le même texte). Il aurait donc été plus judicieux d'utiliser un contrôle *Étiquette*.
Là où les expressions prennent tout leur sens, c'est lorsqu'elles dépendent des valeurs des champs de la source ou d'autres contrôles. Leur valeur variera donc avec les enregistrements, ce ne sont plus des constantes.

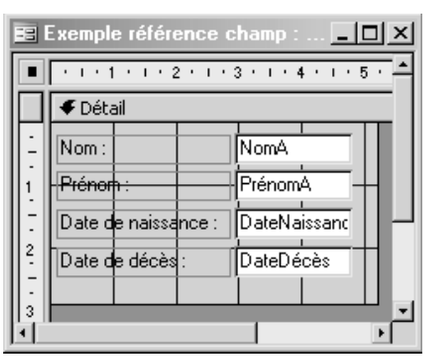
2D. Utilisons les champs et les autres contrôles

2D1. Faisons référence à un champ de la source

Pour faire référence à un champ de la source de données, il suffit de le mettre entre crochets. Nous travaillons sur la table suivante :

Auteur (NumAuteur, Nom, Prénom, DateNaissance, LieuNaissance, DateDécès, LieuDécès, Nationalité)

Voici un exemple de formulaire généré par Access (je ne conserve que le nom et les dates de naissance et décès) :



Je trouve que l'affichage des données fait trop formulaire (!). J'aimerais plutôt regrouper le nom et les dates ainsi :

- le nom : *Stefan Zweig*
- les dates : *(28/11/1881-23/02/1942)*

Et bien, aucun problème : je vais modifier mes contrôles pour qu'ils n'affichent plus les valeurs des champs mais des expressions basées sur ces champs :

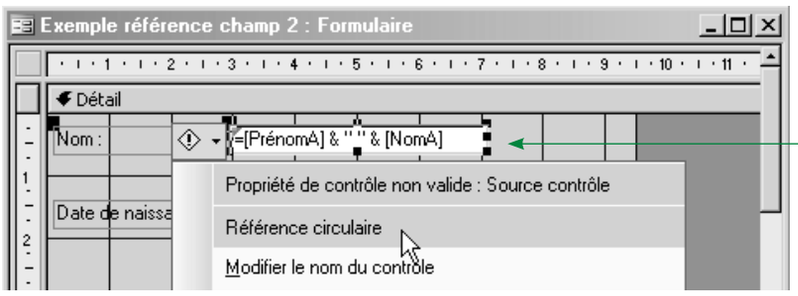
- pour le nom, je concatène le prénom puis un espace puis le nom :

=[PrénomA] & " " & [NomA]

- pour les dates, je les concatène avec les parenthèses et les tirets :

="(" & [DateNaissance] & "-" & [DateDécès] & ")"

Cela donne :

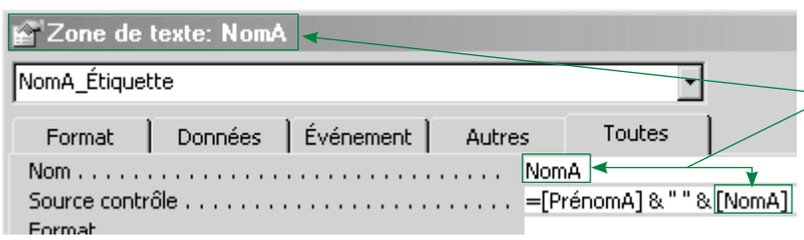


Pour entrer l'expression, soit vous la tapez directement dans la zone de texte, soit vous la saisissez dans sa propriété Source contrôle.

Oups, j'ai un problème de référence circulaire. C'est-à-dire ? En fait, rien de dramatique. Dans mon expression, je fais référence aux champs *PrénomA* et *NomA*. Enfin, cela, c'est mon idée. Car dans mon formulaire, j'ai accès à deux objets *NomA* :

- la zone de texte que je suis en train de définir s'appelle *NomA*. C'était la zone qui, dans la copie d'écran précédente, contenait le nom de l'auteur. Access l'a nommée par défaut du nom du champ lié ;
- le champ *NomA* qui se trouve dans la source de données.

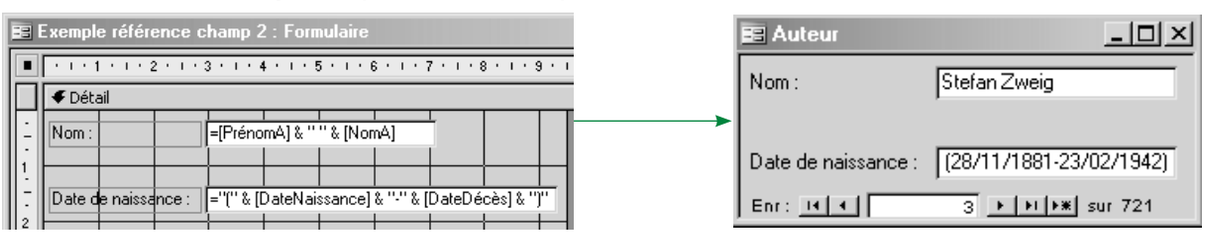
Vous aurez deviné que le contrôle est prioritaire sur le champ. Ainsi, pour Access, le *NomA* utilisé dans l'expression fait référence au contrôle. Et comme nous sommes en train de définir ce contrôle, cela fait une erreur de référence circulaire❶.
Pour fixer les idées, voici une copie d'écran des propriétés :



Voilà le problème ! La valeur du contrôle NomA *est utilisée pour définir la valeur du contrôle* NomA.

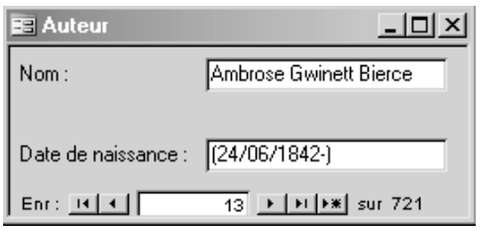
Pour régler ce problème, il suffit de renommer le contrôle. Je modifie sa propriété *Nom* en remplaçant l'ancien nom *NomA* par *NomAuteur*. Et là, plus de problème.
Enfin, plus de problème… Il se pose également pour la zone de texte *DateNaissance*. Je la renomme en *DateNaissanceAuteur*.

Et cette fois, je n'ai plus du tout de problème :



❶ *L'erreur de référence circulaire est classique sous Excel : essayez de définir la formule* =A2+1 *dans la cellule A2.*

Voici la fiche d'un autre auteur :



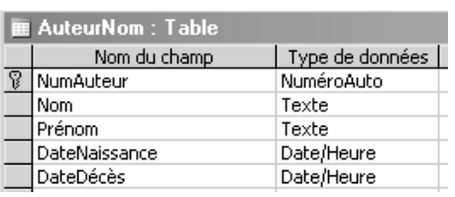
La date de décès d'A. Bierce est inconnue (il a disparu). Du coup, son champ DateDécès *est vide (non renseigné). Cela donne un affichage tout à fait sympathique dans le formulaire.*

Exercice

Testez cela. Provoquez une erreur de référence circulaire et corrigez-la.

2D2. Mauvais nom de champ

Si j'avais défini la propriété représentant le nom de l'auteur en *Nom* au lieu de *NomA*, j'aurais la table suivante :


AuteurNom : Table

	Nom du champ	Type de données
🔑	NumAuteur	NuméroAuto
	Nom	Texte
	Prénom	Texte
	DateNaissance	Date/Heure
	DateDécès	Date/Heure

Par cohérence, j'ai également renommé PrénomA *en* Prénom.

Dans ce cas, le formulaire précédent posera problème :

Oups ! Au lieu de prendre le nom de l'auteur, on a le nom du formulaire.

C'est le même problème qu'avec la récursivité : nous avons deux objets référencés par *Nom* : *Nom* est la propriété contenant le nom du formulaire mais *Nom* est aussi le nom d'un champ de la source de données. Et, encore une fois, la source de données passe en dernier…

Pour régler le problème❶, il faut de nouveau lever l'ambiguïté et dire que le *Nom* auquel je fais référence dans mon expression, c'est le champ *Nom* et pas la propriété *Nom*.

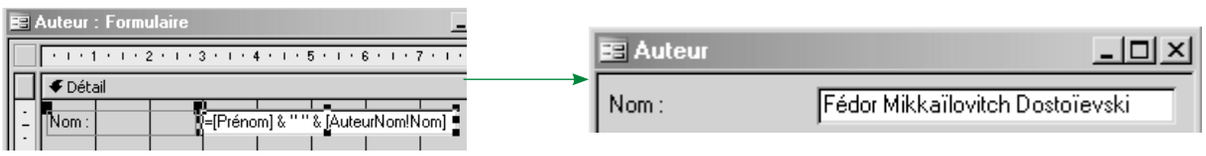
Comment s'en sortir ? Il y a deux solutions :

- on nomme le champ autrement dans la table, par exemple *NomA*. Mais il faudra réécrire les requêtes, les formulaires utilisant ce champ… il aurait fallu y penser dès le MCD !
- on va préciser que le *Nom* dont on parle est le champ *Nom* de la source de données *Auteur* et non la propriété *Nom* contenant le nom du formulaire. Comment faire cela ? Sous SQL, pour lever une ambiguïté d'un champ appartenant à deux tables, on le préfixe par le nom de la table. Ici, on préfixera le champ par le nom de la source de données (soit le nom de la table ou requête alimentant le formulaire). Comme la source du formulaire est *AuteurNom*, j'écris *AuteurNom!Nom*.

❶ *Enfin, la meilleure façon de régler ce problème, ce serait de ne pas l'avoir soulevé en nommant correctement le champ.*

Nous retenons évidemment la seconde solution. Notez bien que j'ai utilisé un point d'exclamation sans espace pour séparer la source et le champ. La valeur de la propriété *Source contrôle* de la zone de texte est =[Prénom] & " " & [AuteurNom!Nom].

Voici le formulaire en mode création et un exemple d'exécution :



En testant cette solution, j'ai obtenu une erreur #Nom ? (indiquant un nom inconnu) dans le contrôle :

Impossible d'avoir le résultat que je voulais, jusqu'au moment où j'ai enregistré, fermé et rouvert le formulaire. Et là, cela a fonctionné. **Donc, en cas d'erreur incompréhensible, quittez et rouvrez !**

Une remarque importante :
Si le formulaire initial permettait la saisie d'un auteur, notre version plus lisible ne permet plus que la consultation (j'aurais donc d'ailleurs dû changer son titre). En effet, je vous rappelle que vous ne pouvez pas saisir la valeur d'une expression. C'est logique, puisqu'une expression se calcule. Sa valeur n'est donc pas arbitraire.

2D3. Faisons référence à un autre contrôle

C'est exactement le même principe : pour utiliser la valeur stockée par un contrôle, on utilisera son nom que l'on notera entre crochets.
Heu, son nom ? Les contrôles ont donc un nom ? Hum. Et c'est quoi, leur nom ? En fait, tout en informatique (variable, objet, formulaire, table…) possède un nom pour pouvoir y accéder. Les contrôles n'échappent pas à la règle. Nous ne nous en étions pas souciés depuis le début de ce cours car Access met des valeurs par défaut. Vous avez d'ailleurs dû le remarquer avec les noms par défaut des requêtes (*Requête1*, *Requête2*…) et des formulaires (*Formulaire1*, *Formulaire2*…). Les contrôles issus de la boîte à outils auront les noms par défaut suivants :

- les zones de texte s'appelleront *Texte1*, *Texte2*…
- les listes déroulantes s'appelleront *Modifiable1*, *Modifiable2*…
- …

Si vous glissez les champs depuis la fenêtre *Liste des champs* pour générer des contrôles, ces derniers porteront le nom du champ.
En tout état de cause, utilisez la propriété *Nom* pour voir ou modifier le nom d'un contrôle. (Elle se trouve sous l'onglet *Autres* de la fenêtre *Propriétés*.)

Un petit exemple ? Envisageons la table suivante :
Facture (NumFacture…, MontantHT, MontantTTC) primaire
Je souhaite saisir dans le formulaire les montants HT et TTC et voir le montant de TVA et le taux correspondant affichés dans deux zones de texte indépendantes❶.

❶ *J'avais présenté ce sujet dans le paragraphe 2A : cela permet de vérifier la saisie.*

Le calcul de la TVA est simple : j'utiliserai les champs *MontantHT* et *MontantTTC* avec l'expression suivante :

=[MontantTTC]-[MontantHT]

(Notez bien que ces champs étant affichés dans des contrôles, je pouvais au choix utiliser le champ ou le contrôle le contenant pour mon expression.)

Je calculerai le taux de TVA ainsi : Taux = (MontantTVA/MontantHT)*100.

J'ai donc besoin du montant de TVA. Or, comme cette valeur n'est pas un champ de la source de données, je dois impérativement faire référence au contrôle indépendant contenant la valeur.
En prenant soin de nommer ce contrôle *MontantTVA*, j'obtiens l'expression suivante :

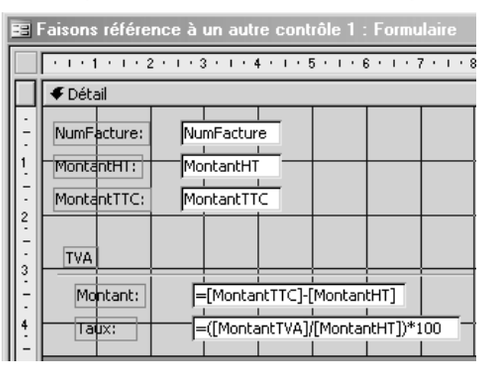
=([MontantTVA]/[MontantHT])*100

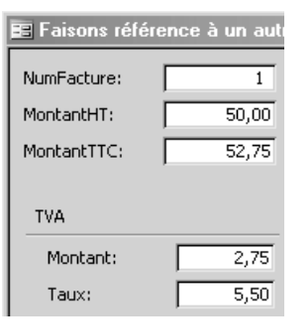
En lisant cette expression, impossible de savoir si je fais référence à des champs ou à des valeurs contenues dans des contrôles. Mais cela n'a aucune importance !

Les copies écran suivantes illustrent cet exemple.

Dans un premier temps, vérifions que la zone de texte contenant le montant de TVA s'appelle *MontantTVA* :

Voyons maintenant le formulaire en mode *Création* (j'ai agrandi les zones indépendantes pour que vous puissiez voir les formules) puis *Formulaire* :

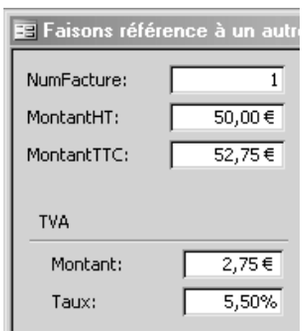




Encore une chose : les unités sont très importantes pour la lisibilité des données affichées (comme sous Excel). Je vais donc attribuer des formats d'affichage adéquats à mes zones de texte (propriété *Format* sous l'onglet *Format*) :

- format *Monétaire (euro)* aux zones de texte contenant les montants HT, TTC et TVA ;
- format *Pourcentage* pour la zone du taux de TVA. Notez que ce format multiplie par cent comme sous Excel. J'enlève donc le « *100 » présent dans ma formule (sinon j'obtiendrais 550,00 %).

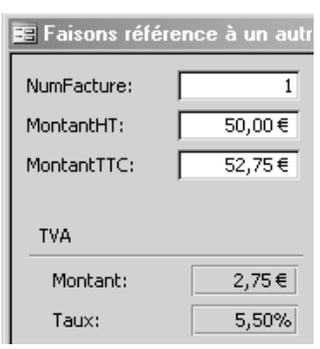
Au final, cela donne :



Je fais cela pour vous montrer les différentes propriétés.
***Mais**, si les propriétés des champs de la table* Facture *avaient été définies correctement, les formats d'affichage utilisés auraient été exploités automatiquement.*

Le format d'affichage est issu d'Excel. C'est une sur-couche par rapport à la valeur. Ainsi, le contenu de *MontantTVA* reste le nombre 2,75 et pas le texte "2,75 €". Les calculs sont donc toujours possibles.

Une dernière chose : pour la sacro-sainte ergonomie que nous étudierons dans la séquence 13, il est bon de distinguer visuellement les zones en lecture seule pour que l'utilisateur comprenne qu'elles ne sont pas modifiables. Par exemple, je donne la valeur *Gravé* à la propriété *Apparence* (onglet *Format*) des zones de texte en lecture seule. Visuellement :



Attention, il s'agit d'être cohérent tout au long de l'application.
Je devrai m'astreindre à utiliser la convention *gravé lecture seule* partout dans l'application (et en parler dans la documentation).

2D4. Allons plus loin

L'expression mise dans la propriété *Source contrôle* peut être complexe (avec des tests ou autre). Essayez de deviner ce que renvoie l'expression suivante avant de lire la suite (cherchez dans l'aide la fonction *VraiFaux)* :

```
=VraiFaux([Civilité]=1;"Monsieur";VraiFaux([Civilité]=2;"Madame";"Mademoiselle"))
```

Solution
La formule dépend de la valeur de *Civilité* (qui est soit un champ de la source de données, soit le nom d'un contrôle du formulaire). Elle renvoie *Monsieur*, *Madame* ou *Mademoiselle* selon que civilité vaut *1*, *2* ou une autre valeur.

Vous noterez en passant que *VraiFaux* est l'exact équivalent de la fonction *si* d'Excel.

Exercice

Testez les différentes expressions que nous avons écrites.

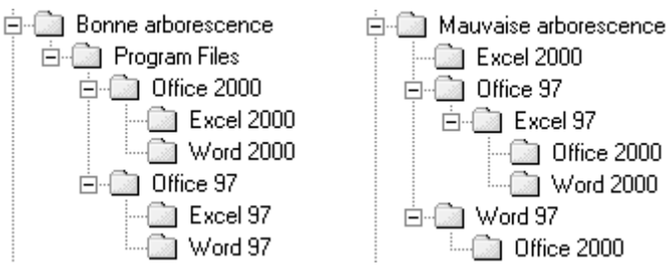
2D5. Allez encore plus loin

Une fois compris le principe des expressions, il est facile et intéressant (utile) d'aller plus loin. En effet, Access propose de nombreuses fonctions intégrées (comme *VraiFaux*) que vous pouvez inclure dans vos expressions.
Pour être exhaustif sur les expressions, consultez l'aide très bien faite (dans la table des matières, choisissez le livre *Expressions* et lisez les différentes rubriques et notamment *Exemples d'expression* [sic] et *Fonctions*).

3. Formulaires et sous-formulaires

3A. Dossiers et sous-dossiers

Faisons un petit rappel de cours sur les dossiers de Windows : un sous-dossier est un dossier contenu dans un autre dossier. Pour avoir un sens, les sous-dossiers et leur contenu doivent posséder un rapport sémantique avec le dossier contenant.
Par exemple :



Vous êtes d'accord que l'arborescence de droite n'a aucun sens, même si on peut la définir (la preuve, je l'ai fait).

3B. Formulaire et sous-formulaire

3B1. Présentation

Le principe est le même que dossier et sous-dossier : un sous-formulaire est un formulaire inclus dans un autre formulaire appelé le formulaire principal.
Pour que cela ait un sens, les données du sous-formulaire doivent avoir un rapport avec les données du formulaire principal. Je vous rappelle que tout formulaire est alimenté par une source de données (sauf les formulaires indépendants). Les sous-formulaires, qui sont avant tout des formulaires, n'échappent pas à la règle.
Il faut donc que la source de données du sous-formulaire dépende de celle du formulaire principal. L'idée sera la suivante : le formulaire principal présente **un** enregistrement de **sa** source et le sous-formulaire présente **tous** (de 0 à n) les enregistrements de **sa** source **qui sont liés** à celui du formulaire principal. Vite, un exemple ! Envisageons le MLD classique de gestion d'une bibliothèque :

Livre (NumLivre, Titre, NbrPages, PrixAchat, **NumAuteur#**)
Auteur (**NumAuteur,** NomAuteur, PrénomAuteur) primaire, étrangère#

Les deux tables *Livre* et *Auteur* vont constituer les deux sources de données. Elles sont liées par un lien clé primaire/clé étrangère sur le champ *NumAuteur*. Ainsi, à un auteur sont associés plusieurs (de 1 à n) livres.

On aura donc un formulaire principal proposant les auteurs et un sous-formulaire affichant les livres de l'auteur courant du formulaire principal.
Retenez bien que tous les enregistrements du sous-formulaire doivent être liés à l'enregistrement courant du formulaire principal. Ainsi, quand l'enregistrement du formulaire principal change, c'est l'ensemble des données du sous-formulaire (donc sa source de données) qui est mis à jour.
En fait, le couple formulaire/sous-formulaire sera toujours utilisé pour représenter les associations [1–n] : à une occurrence de l'entité du côté *?,1* de l'association (affichée dans le formulaire principal) correspond plusieurs occurrences de l'autre entité affichées dans le sous-formulaire. Le lien entre les deux formulaires est géré par Access. Ils doivent donc impérativement être basés sur des tables ou des requêtes possédant un champ de même type sur lequel Access fera une jointure.

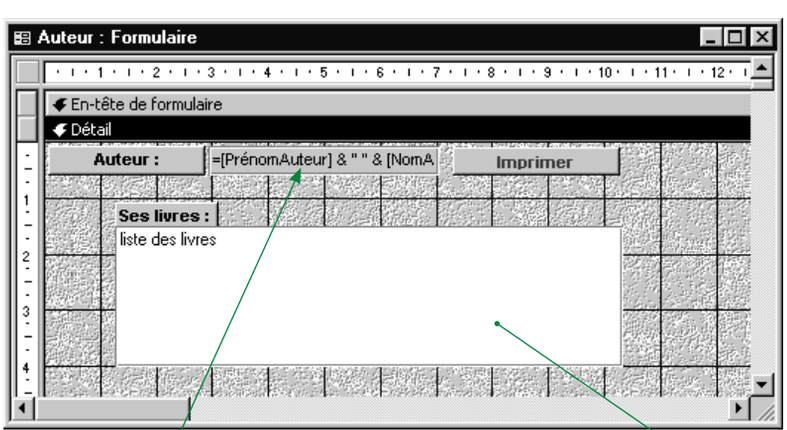
3B2. Exemple

On travaille toujours sur le MLD *Bibliothèque* ci-dessus. Voici un exemple en lecture seule. N'essayez pas de le refaire, vous aurez la main dans la suite du cours.
On va créer deux formulaires, *Auteur* et *Livre*. Le formulaire *Livre* sera sous-formulaire d'*Auteur*. Le lien se fera sur *NumAuteur*, champ liant les deux tables.
Voici le formulaire basé sur la table *Livre* (le numéro d'auteur est présent dans la source de données de ce formulaire mais n'apparaît pas car c'est une variable technique qui n'est pas utile à l'utilisateur). Il est en mode *Feuille de données*.

Encore un vieux formulaire avec des francs...

1677 livres : on a la liste de tous les livres contenus dans la table.

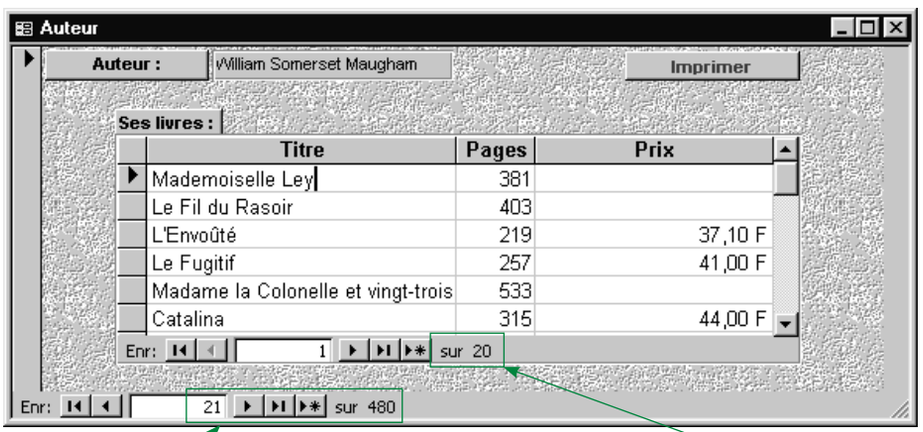
Formulaire *Auteur* en mode *Création* :



On n'affiche pas la valeur d'un champ, mais une expression construite sur deux champs (souci d'ergonomie).

Le formulaire Livre précédent est utilisé en sous-formulaire.

En mode *Formulaire*, on obtient :



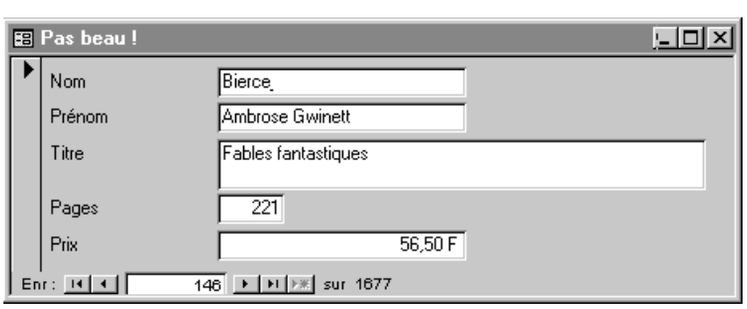
La fiche courante concerne le 21e auteur sur les 480 de la base. Les boutons de navigation permettent de changer d'auteur, le contenu du sous-formulaire Livre *étant mis à jour automatiquement pour toujours présenter les livres de l'auteur courant.*

Cette barre de navigation ne nous donne accès qu'à 20 livres : ceux écrits par W. S. Maugham. On est bien en présence d'une jointure entre les champs NumAuteur *des tables* Livre *et* Auteur.

Cela semble bien compliqué alors que l'on aurait pu créer un seul formulaire basé sur la requête jointure suivante :

```
select NomAuteur, PrénomAuteur, Titre, Pages, Prix
from Auteur A, Livre L
where A.NumAuteur = L.NumAuteur
order by 1 ;
```

Quel formulaire cela donnerait-il ? Le voici :



Cela n'a plus rien à voir : on perd les deux niveaux (on choisit l'auteur et on obtient la liste de ses livres). Ici, on aura linéairement la liste des différents livres avec toutes leurs caractéristiques (pages, auteur, prix…).

3B3. Comment faire un sous-formulaire ?

Je vous l'ai dit dans le paragraphe 3B1 : le lien entre les deux formulaires est géré par Access.

La technique la plus simple est la suivante :

1. Vous créez le formulaire principal comme un formulaire classique en mettant les champs qui vous intéressent. Ce n'est donc qu'un formulaire parmi d'autres à ce moment là.
2. Vous créez le sous-formulaire de la même façon : pour le moment, c'est un formulaire comme un autre où vous mettrez les champs que vous souhaitez.

Je vous rappelle que les sources de données des deux formulaires doivent contenir un champ identique (représentant un lien clé primaire/clé étrangère) sur lequel elles seront synchronisées. Généralement, aucun contrôle ne sera lié à ces champs puisque les identifiants sont techniques donc inutiles pour l'utilisateur.

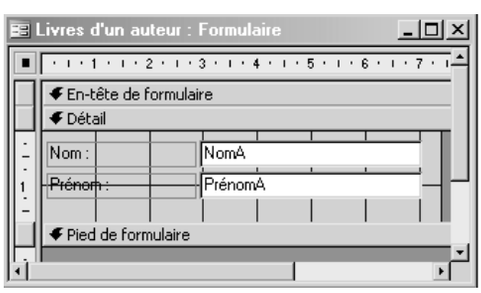
3. Vous ouvrez ensuite le formulaire principal en mode *Création*, vous y glissez depuis la fenêtre *Base de données* celui qui jouera le rôle de sous-formulaire… c'est terminé. Comme vous avez défini les relations entre les clés primaires et les clés étrangères, Access détecte que vous voulez mettre en œuvre un sous-formulaire et paramètre tout.

3B4. Illustration

Première étape

Je crée un formulaire basé sur *Auteur* avec un assistant. Les seuls champs qui m'intéressent sont le nom et le prénom (*NumAuteur* n'est pas affiché mais fait partie de la source donc tout va bien).

mode *Création*



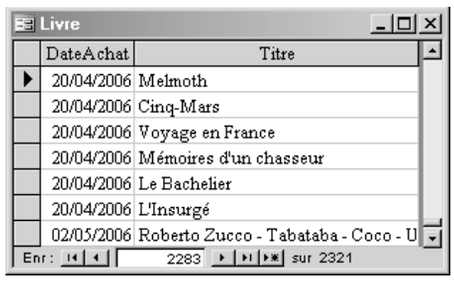
mode *Formulaire*

La **légende** est maladroite puisque je n'affiche que les auteurs. Mais dans quelques instants, ce formulaire contiendra les livres de l'auteur donc tout ira bien.

Deuxième étape

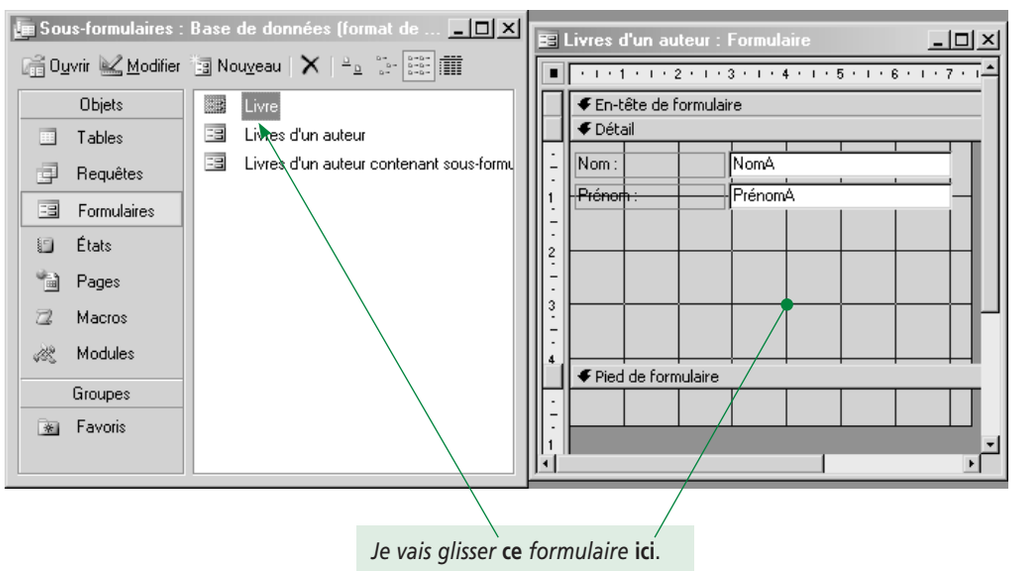
Toujours avec l'assistant, je crée le formulaire basé sur *Livre*. Je ne conserve que la date d'achat et le titre, mais, encore une fois, *NumAuteur* fait partie de la source de données. Notez que je choisis la présentation *Feuille de données* car c'est celle qui est la plus lisible pour un sous-formulaire.

mode *Création* mode *Formulaire*



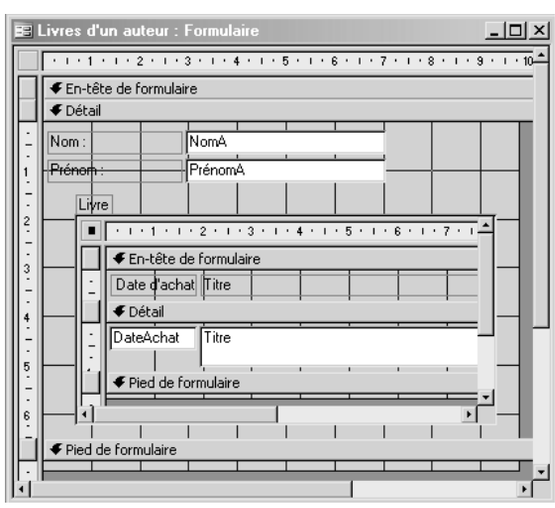
Troisième étape

J'ouvre le formulaire principal en mode *Création* et j'appuie sur *F11* pour avoir la fenêtre *Base de données*. Je sélectionne les objets *Formulaires* puis je glisse le sous-formulaire dans le formulaire *Création*.



Je redimensionne un peu le formulaire et les contrôles et j'obtiens :

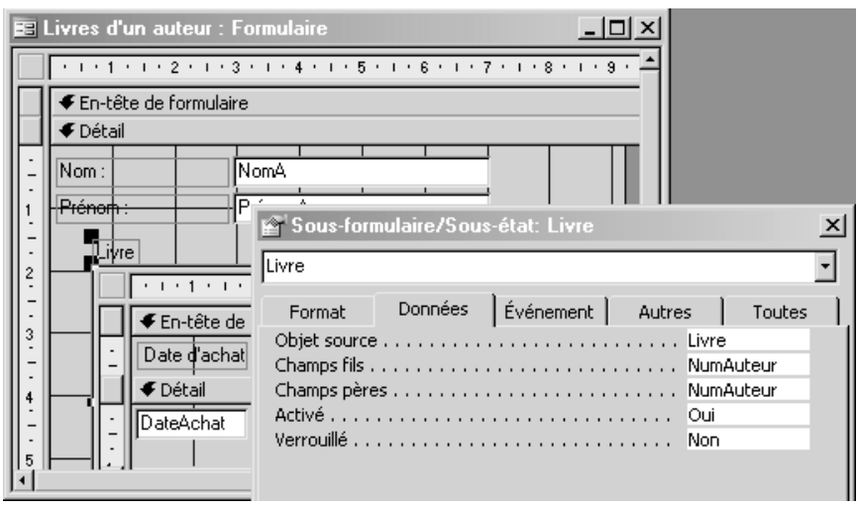
mode *Création* mode *Formulaire*



C'est parfait : les données du sous-formulaire dépendent de celle du formulaire principal.

3B5. Les propriétés du sous-formulaire

Affichons les propriétés du **contrôle** sous-formulaire dans le formulaire principal :



Vous remarquerez les propriétés de ce contrôle (onglet *Données*) :

- *Objet source* est le nom du formulaire utilisé pour jouer le rôle du sous-formulaire ;
- *Champs fils* contient la liste des champs du sous-formulaire permettant de lier ses données avec celles du formulaire principal ;
- *Champs père* contient la liste des champs du formulaire principal permettant de lier ses données avec celles du sous-formulaire.

En clair, les enregistrements des deux formulaires seront synchronisés ainsi :

formulaire principal.champs père = sous-formulaire.champs fils

L'intérêt d'avoir glissé le sous-formulaire dans le formulaire, c'est qu'Access a initialisé ces propriétés. Si vous aviez placé un contrôle sous-formulaire depuis la boîte à outils, vous auriez dû renseigner vous-même ces propriétés (avec l'aide de l'assistant s'il était activé).

3B6. Subtil, subtil… Vous avez dit *subtil* ? Comme c'est astucieux !

Attention à bien comprendre la subtilité : le **contrôle** sous-formulaire **n'est pas** un formulaire. C'est un contrôle qui fait le lien avec un autre formulaire stocké dans la base. Quel sera son comportement visuel ? C'est-à-dire, quelle sera sa représentation graphique lorsque le formulaire sera en mode *Formulaire* ? Pour les autres contrôles, aucun problème : la représentation graphique est la même que l'on soit en mode *Création* ou *Formulaire*.

En revanche, l'aspect visuel du contrôle sous-formulaire en mode *Création* est :

- un rectangle vide sous Access 97 (voir le sous-formulaire *Liste des livres* dans le paragraphe 3B2 précédent. Je l'ai fait sous Access 97) ;
- le formulaire sous-jacent en mode *Création* à partir d'Access 2000 (voir ci-dessus).

On voit ici une des évolutions d'Access : il est plus explicite d'afficher le formulaire plutôt qu'un rectangle. Mais cela vous laisse croire que l'on a physiquement placé le sous-formulaire dans le formulaire principal. J'insiste donc : le contrôle sous-formulaire n'est rien de plus qu'un lien vers le formulaire utilisé (c'est un peu le principe du raccourci sous Windows). Ainsi, lorsque vous modifiez le formulaire jouant le rôle de sous-formulaire, la modification sera prise en compte dans tous les formulaires l'utilisant.

Bien entendu, l'aspect du contrôle sous-formulaire en mode *Formulaire* est le sous-formulaire en mode *Formulaire*❶. On reste cohérent, que l'on soit en mode *Création* ou *Formulaire*.

3B7. Sous-sous-formulaire ?

Peut-on mettre un sous-formulaire dans un sous-formulaire, bref imbriquer des formulaires ?
Par exemple, un formulaire *Client* contenant un sous-formulaire *Facture* lui-même contenant un sous-formulaire *Règlement*. Cela permet d'avoir, pour un client, toutes ses factures et pour chacune de ses factures, le ou les règlements.
On peut imbriquer de nombreux niveaux de formulaires sans problème, bref avoir un formulaire dans un formulaire lui-même dans un formulaire étant dans un formulaire…
Ce qui est techniquement autorisé n'est pas forcément acceptable. Il faut se limiter à trois niveaux (par exemple le client, ses factures et les règlements de ses factures). L'expérience montre que plus d'imbrication est illisible pour l'utilisateur.

3B8. Deux sous-formulaires ?

Vous pouvez également, dans un formulaire, mettre deux ou plus sous-formulaires (qui sont eux au même niveau). Par exemple, le formulaire *Client* peut contenir les sous-formulaires :

- *Facture* (contenant éventuellement un autre sous-formulaire *Règlement*) ;
- *Devis*.

On a visuellement les factures (donc les commandes faites) et les devis pour un client donné.
Attention à ne pas mettre trop de sous-formulaires (imbriqués ou non). En effet, plus il y en a, plus :

- vous faites appel à de nombreuses sources de données et obligez Access à les synchroniser. Cela ralentit l'exécution du formulaire ;
- le nombre d'informations présentes est important, ce qui n'est pas lisible ;
- les liens entre les différentes données sont complexes. L'utilisateur ne s'y retrouve pas.

Exercice

Pour plus d'explications, utilisez l'aide en ligne très explicite. À partir de la table des matières, ouvrez les livres *Création et utilisation de bases de données et d'objets* puis *Objets de base de données* puis *Formulaires* et… régalez-vous !

❶ *N'hésitez pas à relire cette phrase. Elle est élégamment symétrique mais pas forcément claire.*

Synthèse

Nous avons terminé sur les formulaires. Les trois séquences représentent une soixantaine de pages. Est-ce indigeste ? Non, enfin, je l'espère !
J'ai essayé de vous montrer par l'exemple les différents concepts utiles :

- le principe du formulaire ;
- les différents contrôles ;
- les propriétés (formulaires, contrôles) ;
- les zones de texte indépendantes (affichage d'expressions basées sur la valeur d'autres contrôles) ;
- les sous-formulaires.

Vous avez dû comprendre tout cela relativement facilement car, vocabulaire et notions théoriques simples mises à part, il s'agit surtout de recettes et de manœuvres à réaliser de façon automatique. Maintenant, pour tout assimiler et vous dire développeur Access, il faut pratiquer, **pratiquer**, PRA-TI-QUER, à savoir non seulement faire les exercices de ce support, mais aussi développer de petites applications par vous-même.

La principale difficulté de cette séquence est le choix du contrôle adapté à la sémantique de la saisie. Nous avons vu que cela n'avait rien d'évident ! Imaginez un sondage (le recensement est un modèle du genre). Les réponses ne sont pas toutes de même nature. Elles peuvent être :

- une valeur unique parmi une liste (exemple : votre sexe) ;
- plusieurs valeurs parmi une liste (exemple : les équipements de votre domicile) ;
- une valeur que vous devez fournir (exemple : votre âge) ;
- du texte ou un nombre.

Et bien, chaque type de réponse aura une représentation précise dans le document papier. C'est exactement le principe de notre choix de contrôle.
Entraînez-vous à réaliser divers formulaires. Il ne vous manque que l'expérience !
Mine de rien, nous avons tout vu sur les formulaires. Il ne reste plus que l'aspect programmation événementielle que vous connaissez déjà sous VB. Nous l'aborderons dans la séquence 12.
Mais, pour le moment, place au pendant du formulaire : l'état !

Séquence 10

Les états

Dans cette séquence, nous étudierons les états. Ce sera beaucoup plus court que les formulaires car les concepts sont voisins.

▶ *Capacités attendues*

- **Savoir réaliser des états**

▶ *Contenu*

1. Différences entre un formulaire et un état

Rappelons le rôle d'un formulaire :

- afficher des données ;
- permettre la saisie des données.

Pour mener sa mission à bien, le formulaire dispose de toute l'ergonomie de Windows (la souris, les fenêtres et les contrôles). C'est évident puisqu'il s'affiche à l'écran.
Et l'état dans tout cela ? C'est un document destiné à être imprimé. En tant que future feuille de papier, l'état ne permet bien entendu pas de rentrer des données. Il ne permet que d'en afficher. On n'y placera généralement que des étiquettes et des zones de texte. En effet, mettre une zone de liste déroulante est possible, mais cela n'a pas grand sens sur une feuille de papier car l'interactivité est assez limitée sur ce support !
Un état (toujours destiné à être imprimé) permet de restituer de façon lisible :

- les résultats obtenus par une requête ;
- les informations contenues dans une table.

Il existe plusieurs types d'états :

- état instantané en colonne ou tableau ;
- assistant état (entre autres : regroupement/totaux) ;
- assistant graphique ;
- assistant étiquette.

On retrouvera beaucoup de concepts déjà abordés avec les formulaires. Il y a néanmoins une différence importante entre les deux : le formulaire permet une maîtrise complète de l'ensemble de la base par programmation. Enfin, soyons francs… l'état peut être programmé sommairement. Mais il ne possède que peu d'interactivité car, quand il est généré… on le retrouve sur l'imprimante. Impossible de lui ajouter des boutons ou autres contrôles, sauf à l'ouvrir en mode *Aperçu avant impression*. Mais, là encore, on peut se demander l'intérêt de ces contrôles à cet endroit.

Quelques exemples d'états :

- une facture (utilisant les tables *Produit*, *Client* et *Commande*) ;
- la liste de tous les livres de la base classés par auteur ;
- la liste des clients n'ayant pas réglé leur facture à temps ;
- l'édition de lettres de relance ;
- la liste des résultats des différents commerciaux.

En fait, la grande différence entre les formulaires et les états est la suivante :

- un formulaire affiche généralement un enregistrement de la source de données sous-jacente à la fois ;
- un état affichera presque toujours plusieurs enregistrements, soit à la suite (liste des livres de la bibliothèque pour créer le catalogue), soit sur des feuilles distinctes (édition de lettres de relance par exemple).

Un petit lien avec Word : lorsque vous faites du publipostage, vous émulez le fonctionnement d'une base de données générant des états.

2. Description d'un état

2A. Ce qu'il contient

Un état se compose de différentes zones pouvant être supprimées ou modifiées librement. On distingue :

L'en-tête et le pied d'état
Ces zones (en début et en fin d'état) contiendront des informations qui n'apparaîtront qu'une fois, même si votre état se présente sur plusieurs pages. Ce seront habituellement le titre de l'état et des calculs de synthèse.

L'en-tête et le pied de page
Ces zones contiendront des informations qui apparaîtront sur chaque page. Le pied de page contient généralement le numéro de la page.

La zone *Détail*
Cette zone contiendra la description d'une ligne de votre état (ce que l'on affichera de l'enregistrement courant de la source de données).

L'en-tête et le pied d'état ainsi que la zone de détail ne sont pas des nouveautés, nous avons vu l'équivalent avec les formulaires.
Les formulaires n'étaient généralement ouverts qu'en mode *Formulaire unique*. En effet, le formulaire affiche généralement un seul enregistrement à la fois. Pour afficher les autres, il faut cliquer sur les boutons de déplacement. La notion de statistiques ou de résumé n'a donc pas grand sens.
En revanche, l'état imprime d'un coup tous les enregistrements (pas question de gérer le clic sur un document papier !). Il peut couvrir plusieurs pages. En-têtes et pieds sont donc nécessaires :

- pour rappeler le contenu de l'état sur les différentes pages. Par exemple, vous imprimez un tableau sur vingt pages, il faut indiquer l'en-tête des colonnes sur chacune. On utilisera l'en-tête de page ;
- pour avoir des statistiques sur l'état. Vous imprimez toutes les lettres de relance pour les clients qui n'ont pas payé leur facture. Il est bon d'avoir un bilan indiquant combien de relances ont été éditées et les montants moyen et total d'encours que cela représente. Ce sera le bas d'état qui contiendra ces chiffres.

2B. Ses modes d'affichage

Comme le formulaire, l'état peut s'afficher en plusieurs modes :

Mode *Création*
Vous pouvez créer l'état en positionnant les différents contrôles, changer les en-têtes…

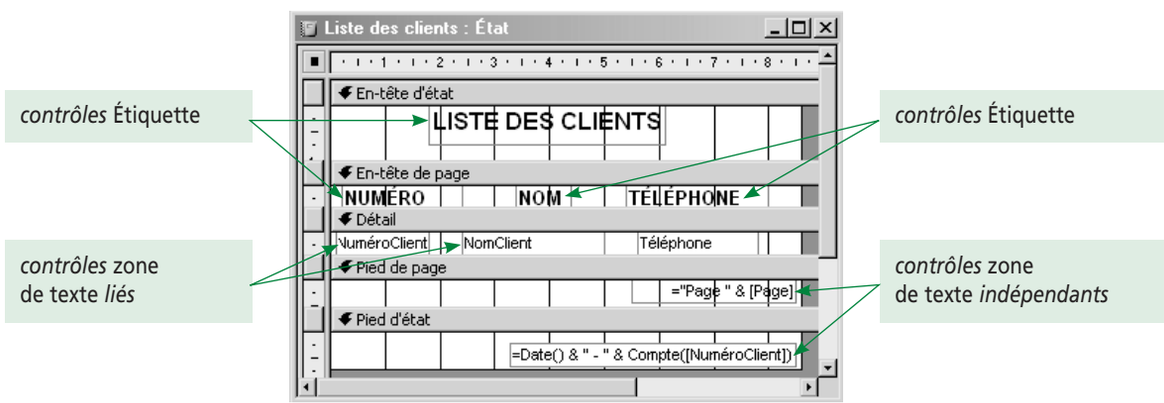
Mode *Aperçu avant impression*
Comme sous Word, vous voyez exactement ce que vous obtiendrez à l'impression. Cela permet de vérifier que tout va bien avant d'imprimer les 500 pages de l'état.

Mode *Aperçu du format*
Ce mode est intéressant. Il permet de vérifier vite fait la mise en forme de votre état sans que la génération soit trop longue. Il n'affichera donc que quelques enregistrements. C'est particulièrement utile si votre état fait 500 pages.

« Mode » *Impression*

Enfin, ce n'est pas un mode d'affichage à proprement parler. Il s'agit de l'impression de l'état. Je vous rappelle que c'est sa finalité ! Les trois autres modes concernent le développeur de l'application. L'utilisateur, lui, n'utilisera que l'impression.

Enfin, de même qu'un formulaire, un état est alimenté par une source de données. Les deux peuvent être vus comme des fonds de page (ou d'écran) où des zones spécifiques sont prévues pour y insérer des données provenant de la source. Comme toujours, la source peut être une table ou une requête. En général, l'état sera alimenté par une requête puisqu'il va présenter des données complexes (issues de l'interconnexion de plusieurs tables). Par exemple, la liste bête et méchante des clients n'a pas vraiment d'intérêt. En revanche, la liste des 100 meilleurs clients avec le montant total du chiffre d'affaires que chacun a généré et la date de leur dernière commande, c'est intéressant.
Voici un exemple d'état en mode *Création* (pour le coup, il est basé sur une table *Client* classique).



Vous noterez que mon état n'est constitué que de deux types de contrôles : des intitulés et des zones de texte. En pratique, seuls ces contrôles seront utilisés puisqu'ils permettent d'afficher du texte simplement. Notez que les zones de texte permettent d'afficher des champs de la source de données ou des expressions comme dans les formulaires.

Voyons maintenant comment cet état va se présenter en mode exécution (donc à l'impression).
En passant en mode *Aperçu avant impression*, on obtient la première page :

LISTE DES CLIENTS

NUMÉRO	NOM	TÉLÉPHONE
22	Teckel sympa	48 95 45 55 24
17	Nina le petit teckel	12 45 45 62 56
34	Pollen	13 18 61 86 18
	…	
11	Engeance alfique	11 86 17 61 71
58	Petite Chose 1	25 71 17 17 17
55	Petite Chose 2	41 75 11 71 17

Page 1

Voici maintenant les deuxième et dernière pages :

deuxième page

NUMÉRO	NOM	TÉLÉPHONE
35	Bipsie	53 69 88 41 20
432	Niok	38 46 35 26 77
	…	
98	Tonus	38 47 66 00 37
112	Démon	95 83 57 33 38
512	Pétula	99 04 59 33 41
47	Zoé	22 54 77 96 30

Page 2

dernière page

NUMÉRO	NOM	TÉLÉPHONE
95	Élan	82 11 47 88 99
103	Pépin	74 84 55 96 55
213	Raoul	64 89 89 65 55
115	Froggy	55 66 57 96 55

03/11/2007 – 503 clients

Page 50

Je suis désolé pour ces modestes dessins, mais je peux difficilement faire des copies d'écran…
Que peut-on noter à la vue de ces pages ?

- la dernière n'est pas forcément pleine… bah oui, si l'on a épuisé la source de données, on ne va pas en inventer !
- vous devez maintenant comprendre le fonctionnement des en-têtes et pieds de page et d'état ;
- notez (sur la dernière page) que le pied d'état se trouve avant le pied de page, tandis que l'en-tête d'état est avant l'en-tête de page (première feuille) ;
- le pied d'état donne ici la date du jour et le nombre de clients ; cela vient d'une expression dans la zone de texte. Attention, la date en question n'est pas celle de la conception de l'état (cela, on s'en moque) mais la date du jour de la visualisation donc de l'impression. C'est utile pour dater les factures, les relances…

Une dernière remarque qui a son importance
L'état présenté ici possède 50 pages. Comme les clients sont sans ordre particulier, je vous souhaite bien du courage pour en trouver un. Dans la vraie vie, on doit impérativement avoir les clients triés dans un ordre précis (sur le numéro ou le nom selon les besoins).

Ainsi, comme je l'ai dit précédemment, nous n'allons pas baser l'état sur la table *Client* mais sur une requête renvoyant les enregistrements de *Client* triés. Par exemple :

```
select *
from client
order by NomClient ;
```

Qu'allons-nous voir dans la suite de cette séquence ? Et bien, les toutes petites choses propres aux états et que nous n'avons donc pas abordées dans les séquences sur les formulaires.

3. Les différents types d'états

Pour créer un état, sélectionnez les objets *États* dans la fenêtre *Base de données* puis, comme avec les formulaires, utilisez un assistant ou faites tout vous-même. Vous aurez d'ailleurs les mêmes outils : les fenêtres *Propriétés* et *Liste des champs* et la *boîte à outils*.
La notion de sous-formulaire se systématise dans les états avec le regroupement. L'état affichant de nombreux enregistrements à la fois (alors que le formulaire n'en affiche qu'un sauf en mode *Feuille de données*), on peut regrouper les enregistrements par champs.
Par exemple, je vais éditer la liste de tous mes livres en les regroupant par auteur : j'aurai tous les livres de Balzac puis tous ceux de James, puis tous ceux de Zola… la liste sera classée par auteur puis par titre de livre.

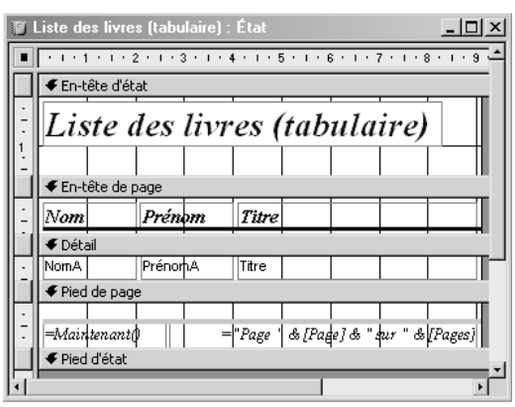
Parce que c'est généralement nécessaire, les états avec regroupement proposent également des calculs d'agrégat sur les différents groupes. Je peux donc avoir la liste des livres, leur nombre et leur prix moyen, le tout par auteur.

Nous allons survoler les assistants basiques et approfondir les états avec regroupement.

3A. *État instantané tabulaire (en tableau)*

Un état instantané en tableau présente chaque enregistrement sur une ligne, les noms des champs étant répétés sur chaque page. Par exemple :

Mode *Création*



Mode *Aperçu avant impression*

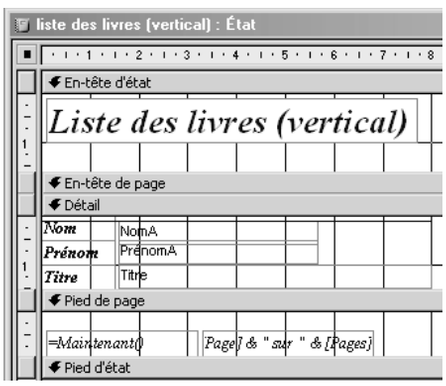
Liste des livres (tabulaire)

Nom	*Prénom*	*Titre*
Abbott	Edwin A.	Flatland
Abbott	Edwin A.	Flatland
Abé	Kôbô	Mort anonyme et autres nouvelles
Ackroyd	Peter	Un puritain au paradis
Adams	Douglas	Beau comme un aéroport
Adams	Douglas	Globalement inoffensive
Adams	Douglas	La Vie, l'Univers et le Reste
Adams	Douglas	Le Dernier Restaurant avant la Fin du
Adams	Douglas	Le Guide du Routard Galactique
Adams	Douglas	Salut, et encore merci pour le poisson
Adams	Douglas	Un cheval dans la salle de bain
Adler	Laure	La Vie quotidienne dans les Maisons
Aguéev	M.	Roman avec cocaïne
Aldiss	Brian W.	Barbe-grise

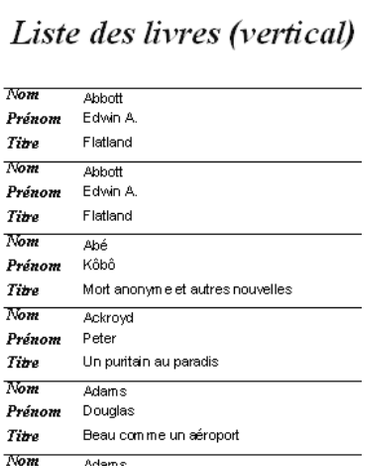
3B. État instantané en colonnes (vertical)

Un état en colonnes affiche les enregistrements sur une colonne (un champ par ligne).

Mode *Création*



Mode *Aperçu avant impression*



Liste des livres (vertical)

Nom	Abbott
Prénom	Edwin A.
Titre	Flatland
Nom	Abbott
Prénom	Edwin A.
Titre	Flatland
Nom	Abé
Prénom	Kôbô
Titre	Mort anonyme et autres nouvelles
Nom	Ackroyd
Prénom	Peter
Titre	Un puritain au paradis
Nom	Adams
Prénom	Douglas
Titre	Beau comme un aéroport
Nom	Adams

Avez-vous remarqué que les enregistrements présentés n'étaient pas vraiment dans un ordre aléatoire ? Il semblerait bien que quelqu'un nous les ait triés ! Et bien, c'est moi. Enfin… à un moment donné, l'assistant m'a demandé si je voulais trier mes enregistrements et j'ai dit oui (par nom, puis par prénom puis par titre). Nous y reviendrons.

3C. Assistant état : mise en œuvre du regroupement

3C1. Un exemple en guise d'introduction

Considérons la table *Client (Numéro, Nom, Tél, Département)* d'une entreprise de télémarketing. La propriété *Département* contient les deux chiffres du département (54 pour Nancy).

L'entreprise souhaite obtenir la liste de ses clients par département afin de réorganiser son équipe de commerciaux. Voici la requête SQL permettant d'avoir les clients (nom et téléphone) pour chaque département. Bien entendu, pour un département donné, les clients seront présentés par ordre alphabétique. Les départements eux-mêmes seront triés.

```
select *
from Client
order by Département, Nom ;
```

*(Je me suis simplifié le travail en mettant * dans la clause* select. *Je récupère donc le champ* Numéro *dont je n'ai que faire… pas de problème, je ne l'utiliserai pas dans l'état.)*

Si l'on fait un état tableau basé sur cette requête, on obtiendra ce qui suit :

LISTE DES CLIENTS PAR DÉPARTEMENT

DÉPARTEMENT	NOM	TÉLÉPHONE
21	Durand S.A.	03 12 87 95 66
21	FNAC	03 97 45 86 54
45	Engeance alfique	11 86 17 61 71
45	Nina le petit teckel	12 45 45 62 56
45	Teckel sympa	48 95 45 55 24
54	La Redoute	02 08 78 96 54
54	SNCF	01 02 00 55 78
59	Dupond	02 01 47 89 87
59	Petite Chose 2	41 75 11 71 17
59	Peugeot	05 14 78 96 57
95	Ma tante	01 43 21 65 42
95	Petite Chose 1	25 71 17 17 17
95	Pollen	13 18 61 86 18

Ce n'est pas très joli ! Si l'on fait un état à partir de l'assistant état, on pourra très simplement obtenir ce qui suit en mettant en œuvre le regroupement :

LISTE DES CLIENTS PAR DÉPARTEMENT

DÉPARTEMENT	NOM	TÉLÉPHONE
21		
	Durand S.A.	03 12 87 95 66
	FNAC	03 97 45 86 54
45		
	Engeance alfique	11 86 17 61 71
	Nina le petit teckel	12 45 45 62 56
	Teckel sympa	48 95 45 55 24
54		
	La Redoute	02 08 78 96 54
	SNCF	01 02 00 55 78
59		
	Dupond	02 01 47 89 87
	Petite Chose 2	41 75 11 71 17
	Peugeot	05 14 78 96 57
95		
	Ma tante	01 43 21 65 42
	Petite Chose 1	25 71 17 17 17
	Pollen	13 18 61 86 18

(Je n'ai inclus aucun calcul de cumul.)

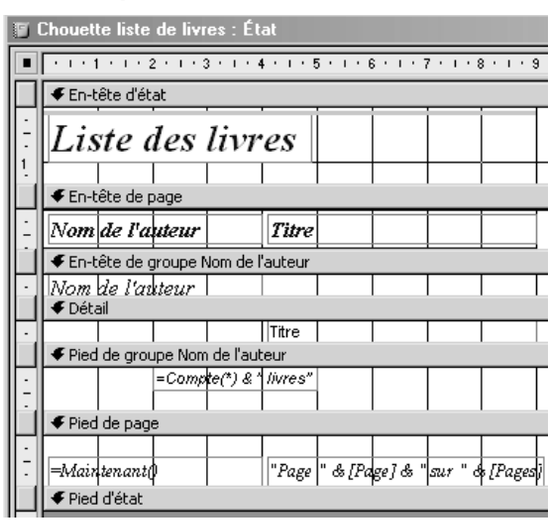
3C2. Détaillons le regroupement

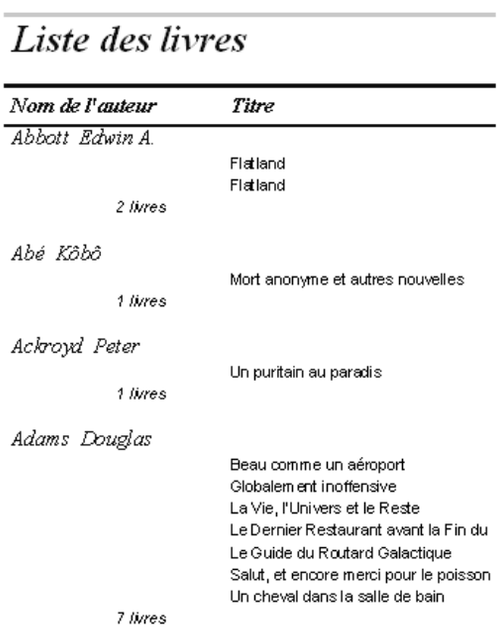
L'assistant état permet entre autre de réaliser un état avec *Regroupement/Totaux* : les données sont regroupées selon la valeur d'un ou plusieurs champs. On retrouve la notion de *group by* des requêtes SQL, mais également ce que nous avons vu avec les sous-formulaires.
Regardez l'état ci-dessus : on imagine aisément le formulaire principal contenant le département et le sous-formulaire affichant les clients de **ce** département. La différence ? Le formulaire affiche un département (enregistrement) à la fois alors que l'état les affiche tous.

Revoyez le paragraphe 3B2 de la séquence 9 : je vous montrais un sous-formulaire présentant les livres d'un auteur donné. Cela, c'est le principe de l'état avec regroupement. Nous comparions ce formulaire avec un autre basé sur une requête jointure. On obtenait alors quelque chose de très similaire à mes deux états précédents (3A et 3B). Ils présentent tous les livres avec leurs auteurs, mais de façon linéaire.

Notre objectif est maintenant de faire des groupes (paquets) de livres : un groupe par auteur. Access gérant la notion de groupe (le sous-formulaire contient un groupe d'enregistrements dépendant de celui du formulaire principal), il pourra ensuite faire tous les calculs d'agrégat que je veux sur chaque groupe. Exemple ? Je pourrai obtenir le nombre de livres par auteur, le prix moyen des livres d'un auteur… Ce sont les fonctions d'agrégat de SQL en plus évoluées car toutes les fonctions d'un langage de programmation sont accessibles.

Par exemple :





Liste des livres

Nom de l'auteur	*Titre*
Abbott Edwin A.	
	Flatland
	Flatland
2 livres	
Abé Kôbô	
	Mort anonyme et autres nouvelles
1 livres	
Ackroyd Peter	
	Un puritain au paradis
1 livres	
Adams Douglas	
	Beau comme un aéroport
	Globalement inoffensive
	La Vie, l'Univers et le Reste
	Le Dernier Restaurant avant la Fin du
	Le Guide du Routard Galactique
	Salut, et encore merci pour le poisson
	Un cheval dans la salle de bain
7 livres	

Nous allons bientôt détailler l'assistant de génération d'un état. Mais avant cela, il est intéressant de bien étudier l'exemple ci-dessus. J'ai basé l'état sur la requête suivante :

```
SELECT NomA & " " & PrénomA AS [Nom de l'auteur], Titre
FROM Auteur AS A, Livre AS L
WHERE A.NumAuteur = L.NumAuteur;
```

J'ai effectué un regroupement sur le champ *Nom de l'auteur*. Dans l'état en mode *Création*, vous voyez trois zones liées à ce groupe : *en-tête de groupe*, *Détail* et *pied de groupe*. Pour chaque auteur (chaque enregistrement de la source de données *Auteur*), on aura donc :

- un en-tête : j'y mets le nom de l'auteur ;
- la zone *Détail* : elle contiendra tous les livres de l'auteur ;
- un pied : j'y mets le nombre de livres de l'auteur ; notez que *livre* est au pluriel même si l'on n'en a qu'un. Cela ne me plait pas. Nous réglerons le problème en 3C4.

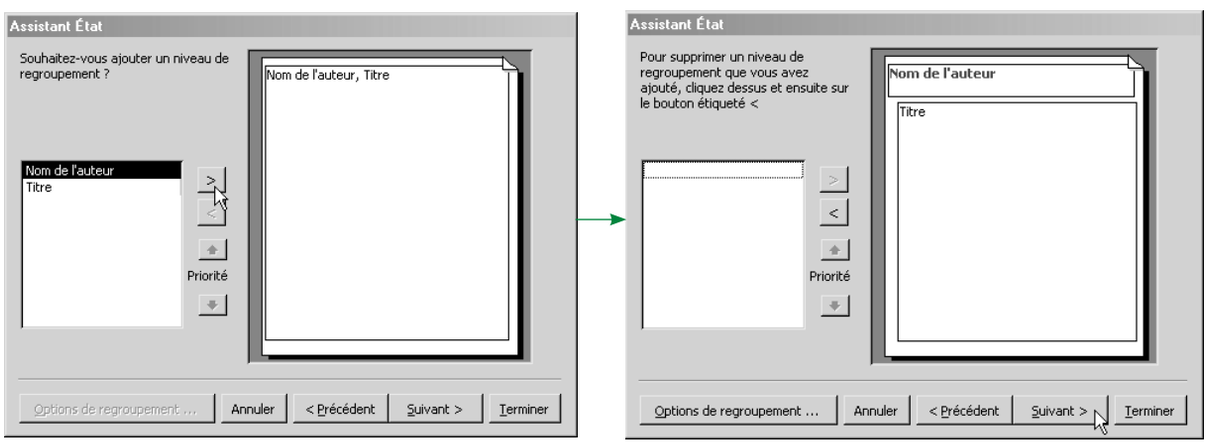
Ces notions d'en-tête et de pied de groupe contenant des données de synthèse représentatives du groupe sont présentes dans les formulaires (principaux). Elles ne sont pour autant pas explicites comme dans l'état car leur usage y est beaucoup moins systématique.

3C3. Utilisons l'assistant

Nous allons utiliser l'assistant pour refaire cet état. Avant de commencer, écrivez et enregistrez la requête du paragraphe précédent.

On y va : sélectionnez les objets *États* puis *Nouveau*. Vous obtenez alors les écrans suivants :

1. Choisissez *Assistant état* et basez-le sur la requête précédente.
2. Sélectionnez les deux champs (*Nom de l'auteur* et *Titre*).
3. C'est ici que vous choisissez les éventuels regroupements. Vous choisissez dans la partie gauche de la fenêtre et observez le résultat à droite. Faites des essais, faites un regroupement sur le nom de l'auteur et allez voir les différentes options de regroupement. Voici une illustration de la fenêtre ci-dessous.



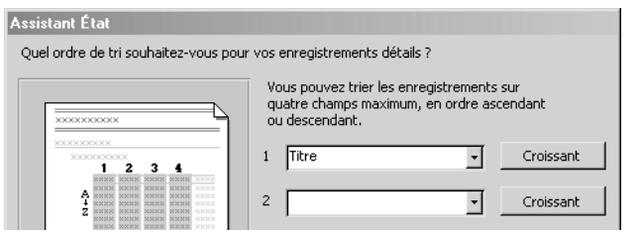
Notez bien les choix de regroupement : *par Livre* ou *par Auteur*. *Livre* et *Auteur* sont les noms des deux tables utilisées dans la requête. Regrouper *par Auteur*, donc d'après les enregistrements de cette table, donne bien ce que l'on veut : pour un auteur, tous ses livres.

Au contraire, regrouper *par Livre* donnera une liste linéaire sans regroupement (il n'y a qu'un auteur par titre de livre d'après le MLD).

Au fait… comment Access le sait-il, que le regroupement est possible par auteur mais pas par livre ? Et bien, c'est parce que j'ai défini une relation entre les clés primaire et étrangère des deux tables : Access sait donc qu'à un auteur sont associés plusieurs livres mais qu'à un livre ne correspond qu'un auteur. Ah ah ! Les relations sont vraiment utiles !

Je vous conseille vivement d'utiliser le bouton *Afficher plus d'informations* et de lire l'aide bien faite.

4. Ordre de tri : vous pouvez trier les enregistrements de la zone *Détail*, donc les livres. Access ne propose pas de trier sur le nom car c'est automatique : les regroupements sont toujours triés. Vous noterez le bouton à côté du champ : on peut trier par ordre croissant ou décroissant. Moi, je trie par titre croissant.



5. Les boîtes de dialogue suivantes vous posent quelques questions de style. Je vous laisse le plaisir de la découverte. N'hésitez pas à activer l'aide dans la dernière !

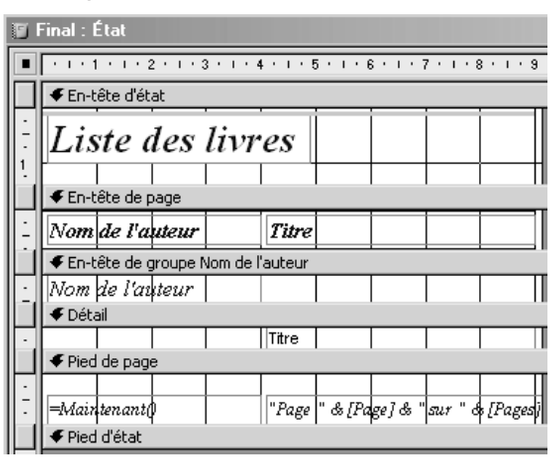
3C4. Ajoutons des calculs d'agrégat

Observez l'état que nous venons de créer. Il est bien, mais enfin… il manque d'informations de synthèse. J'aimerais par exemple savoir combien je possède de livres par auteur. L'état du paragraphe 3C2 donnait cette information.

Nous allons maintenant le faire. Le principe est simple : comment faisait-on pour afficher des résultats de calcul d'expression dans les formulaires ? On utilisait des contrôles zones de texte, la formule de l'expression à calculer étant mise dans la propriété *Source contrôle*.

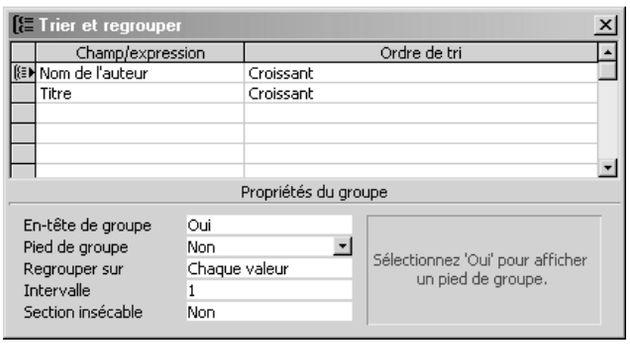
Nous allons faire la même chose. La seule nouveauté est de connaître les fonctions d'Access permettant de calculer ce que l'on veut. Et pour cela, pas de miracle… il faut consulter l'aide. Les calculs d'agrégat se font sur un agrégat de valeurs. Par exemple, le calcul du nombre de livres se fait sur un ensemble de livres (ceux d'un auteur donné). Ainsi, comme nous l'avons vu dans le paragraphe 3C2, les fonctions d'agrégat seront mises dans l'en-tête ou le pied d'un regroupement (sachant qu'elles s'appliquent aux enregistrements qui sont dans la zone *Détail*). Comprenez bien cela car c'est implicite dans Access.

Revenons au formulaire fait précédemment avec l'assistant.



Je veux donc mettre le nombre de livres par auteur dans le pied de groupe *Nom de l'auteur* comme dans le formulaire du paragraphe 3C2. Mais… mince alors ! Je vois bien l'en-tête et le détail du groupe mais pas le pied. Que faire ?

Par défaut, le pied n'est pas affiché car l'assistant n'y met rien. Pour le voir, menu *Affichage/Trier et grouper*. Cette commande ouvre une boîte de dialogue suivante❶ :

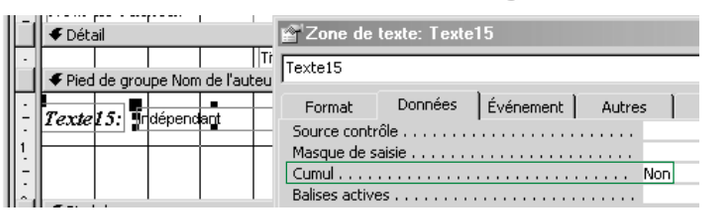


On voit tous les regroupements et leur paramétrage. Vous voyez que l'on regroupe selon le champ *Nom de l'auteur* par ordre croissant puis par *Titre* croissant. Cela appelle deux remarques :

1. Le tri sur nom de l'auteur est fait sans vous demander votre avis. Je l'ai déjà dit précédemment : regrouper sans trier n'ayant pas de sens, l'assistant trie de lui-même.
2. Il semblerait que l'on regroupe sur le titre. Or, on n'a rien demandé de tel. Que se passe-t-il ? En fait, *Titre* est présent ici pour indiquer que l'on trie les enregistrements sur ce champ. Regrouper sur des valeurs toutes distinctes ne change rien.

Avant de fermer la fenêtre *Trier et regrouper*, prenez le temps d'étudier avec l'aide les différentes propriétés.

Placez maintenant une zone de texte dans le pied de groupe (si vous n'avez pas accès à la boîte à outils, menu *Affichage/Boîte à outils*). Affichez la fenêtre *Propriétés* de cette zone de texte et sélectionnez l'onglet *Données*. Vous devez obtenir ceci :



Notez la propriété Cumul. *On ne la trouve que dans l'état. Elle permet d'indiquer si l'on remet les compteurs à zéro pour chaque groupe (donc ici pour chaque auteur). Comme c'est le cas, on ne cumule pas. (D'où la valeur* Non.*)*

Exercice

Pour plus d'explications, utilisez l'aide en ligne très explicite. À partir de la table des matières, ouvrez les livres *Création et utilisation de bases de données et d'objets* puis *Objets de base de données* puis *États* et… régalez-vous !
Puis, à partir de la table des matières, ouvrez les livres *Utilisation des données* puis *Recherche, tri et regroupement des données* puis *Groupement des données* puis lisez la rubrique *Exemples d'états regroupés* et… régalez-vous ! (Gourmand, va)

Ce qui m'intéresse, c'est ce que l'on va mettre dans *Source contrôle*. Il nous faut une expression comptant le nombre de livres (d'enregistrements) situés dans la zone *Détail*.

Dans la séquence précédente (paragraphe 2D5), je vous avais renvoyé à l'aide pour

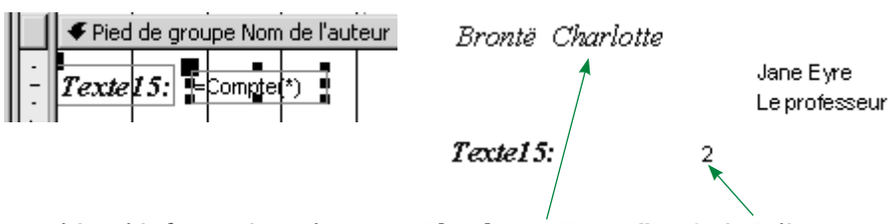
❶ *Ah… un bug ; mineur, mais un bug quand même : la commande est* Trier et grouper *alors que la fenêtre s'appelle* Trier et **re**grouper.

approfondir les expressions. Si vous n'y êtes toujours pas allé, il est temps ! Allez consulter dans l'aide, à partir de la table des matières, les livres :

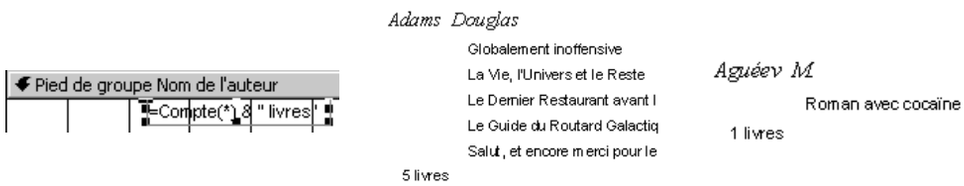
- *Expressions* pour une aide générale sur les expressions. Cela est utile pour les formulaires et les états ;
- *Expressions* et lisez les différentes rubriques et notamment *Exemples d'expression* [sic] *et Fonctions*.

Pour notre propos, l'aide propose une technique intéressante (voir *Utilisation d'états/Calcul de totaux et utilisation d'expressions/Numérotation et comptage des enregistrements/Compter le nombre d'enregistrements dans chaque groupe d'un état*). L'intérêt de cette méthode est qu'elle évite l'emploi d'une fonction. L'inconvénient, c'est qu'elle utilise deux contrôles. Étudiez-la néanmoins !

Ma technique consiste à utiliser la fonction *Compte* (équivalent à la fonction d'agrégat *Count* de SQL). Comme on veut compter les enregistrements *Livre*, on mettra au choix =Compte(*) ou =Compte(Titre) dans *Source contrôle* de la zone de texte.
Faites-le et testez. Cela donne :



Nous avons bien l'information : l'auteur **Charlotte Brontë** a écrit **2** livres ; enfin, plus précisément, j'en possède 2 de cette auteure.
Réglons maintenant un problème d'ergonomie. L'étiquette de la zone (*Texte15* dans mon cas) n'est évidemment pas acceptable. On pourrait mettre par exemple *Nombre de livres*. Mais je vais faire autrement : je vais supprimer l'étiquette (cliquez dessus, puis *Suppr*).
L'affichage que je veux obtenir est « *4 livres* ». Je vais donc concaténer le nombre de livres obtenu par *Compte(*)* avec *livres*.
J'obtiens alors l'expression = Compte(*) & " livres". Visuellement, on obtient ce qui suit.



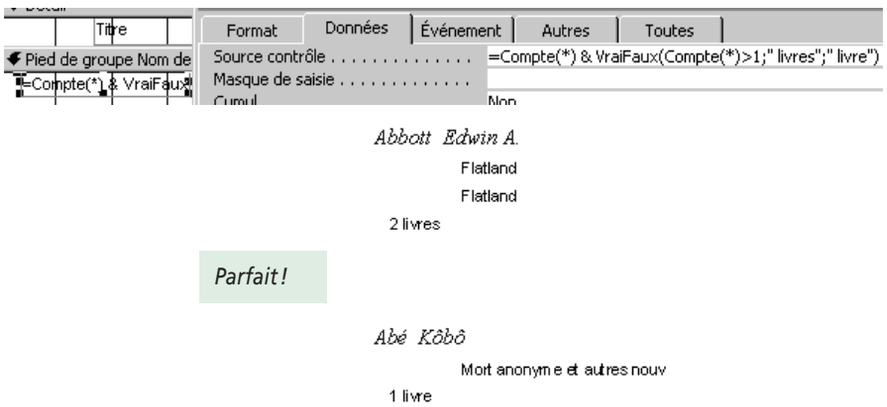
Voyez le cas de M. Aguéev : je ne possède qu'un de ses livres. Dans l'état, c'est écrit *1 livres*. Je vous le dit tout net, cela me gène (j'ai déjà évoqué le problème dans le paragraphe 3C2).
Je souhaite écrire *livre* s'il n'y en a qu'un et *livres* sinon. Comment faire ? Je vais simplement utiliser la fonction *VraiFaux* (si cela ne vous dit rien, vous n'êtes pas allé dans l'aide comme je vous l'avais demandé paragraphe 2D4 de la séquence précédente…).
C'est la même fonction que le *si* d'Excel. La syntaxe est sans surprise :

VraiFaux (booléen ; valeur_si_vrai ; valeur_si_faux)

Mon expression devient :

=compte(*) & VraiFaux(compte(*)>1 ; " livres" ; " livre")

Vérifions :



En exploitant le fait que *valeur_si_faux* est facultatif, on peut faire plus concis :

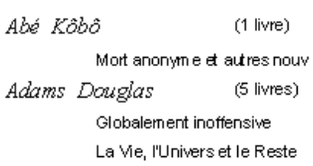
=compte(*) & " livre" & VraiFaux(compte(*)>1 ; "s")

(Si j'ai plus d'un livre, je rajoute un *s* au mot *livre*.)

Dans la propriété *Source contrôle* de la zone de texte, mettez l'une des expressions, testez, puis mettez l'autre. Vérifiez qu'elles sont bien équivalentes.
Il ne fallait pas se passer de cette optimisation : elle prend deux minutes et le résultat obtenu fait beaucoup moins *traitement automatique*.

3D. Allons plus loin

Enfin, plus loin… on a fait le tour de ce qu'il y avait à dire ! Pour modifier un état, il suffit d'appliquer ce que l'on connaît déjà sur les formulaires. Pour connaître d'autres modes ou fonctions de cumul, consultez l'aide.
Pour s'amuser, essayez de partir de l'état du paragraphe précédent et d'obtenir le résultat suivant :



Une petite aide : il suffit de déplacer (par *couper/coller*) la zone de texte concernée, de lui ajouter les parenthèses par concaténation (à savoir ="(" & "Compte..." & ")") et de supprimer le pied de groupe devenu inutile.

Synthèse

Il faut retenir le rôle de l'état et les différents types qui existent, avec ou sans regroupement. On peut utiliser un état dans deux cas très différents :

- pour imprimer un document lié à un enregistrement précis d'une table. Par exemple, on imprime une facture, une relance, un bon de commande… Ce cas est trivial car il ne met en œuvre aucun concept spécifique à l'état. On retrouve en fait un formulaire imprimé avec des zones de texte comme seuls contrôles ;
- pour obtenir un document papier contenant plusieurs enregistrements (liste des clients, des factures impayées, liste des opérations comptables…). On exploitera alors les différentes fonctionnalités de regroupement et de calcul de synthèse.

L'état est donc très proche du formulaire, à la différence fondamentale que celui-ci affiche un enregistrement à la fois alors que celui-là en affiche plusieurs. Ainsi :

- le formulaire classique affichera un client et permettra de passer à d'autres avec des boutons de déplacement. L'état correspondant contiendra d'emblée la liste des clients ;
- le formulaire principal affichera un client (boutons de déplacement pour en changer) et son sous-formulaire donnera toutes ses factures. L'état correspondant sera un état avec regroupement sur le client. Il contiendra tous les clients avec, pour chacun d'eux, la liste de toutes ses factures.

Finalement, le formulaire est une fenêtre sur une liste : pour voir cette dernière, il faut déplacer la fenêtre (sous-entendu changer d'enregistrement courant). L'état, lui, affiche cette liste entièrement.
La notion de groupe est implicite dans les formulaires (mais elle existe puisqu'un sous-formulaire est un groupe). Elle est en revanche explicite avec des en-tête et pied de groupes dans les états car, ceux-ci imprimant généralement beaucoup plus de données qu'un formulaire, on a besoin de cette hiérarchisation.
Par exemple, si mon petit teckel Nina était proviseure, elle pourrait :

- consulter à l'écran (formulaire) les élèves d'une classe afin d'avoir le téléphone des parents de l'un d'eux ;
- imprimer un état pour avoir l'annuaire du lycée (la liste de tous les élèves avec le téléphone des parents). Cet état posséderait trois groupes :
  1. les niveaux (secondes, premières, terminales),
  2. pour chaque niveau, les différentes classes (terminale S, STG…),
  3. dans chaque classe, les élèves triés par ordre alphabétique.

En passant, une question piège : chaque professeur principal réclame à teckel proviseur la liste des élèves de sa classe. Comment teck' prov' va-t-il les imprimer ? Faudra-t-il un état par classe ? Non ! Il reprendra l'état ci-dessus en plaçant un saut de page après chaque classe ; chacune sera donc sur une feuille distincte.
Teckel en profitera sans doute pour changer un peu l'en-tête de page et supprimer l'en-tête d'état devenu inutile. En effet, l'état ne sera plus un document, mais une suite de feuilles à distribuer. On retrouve les deux types d'états mentionnés au début de la synthèse.

Un autre exemple ? Et bien, Raoul le petit documentaliste sympa consultera à l'écran la notice d'un ouvrage de son fond mais imprimera la liste de tous les ouvrages.

Séquence 11

Les macros

L'étude des macros sera brève car c'est un outil obsolète, remplacé par VBA. Il est néanmoins important de savoir ce que c'est pour pouvoir travailler sur d'anciennes bases Access lors de travaux de maintenance ou de migration.

▶ *Capacités attendues*

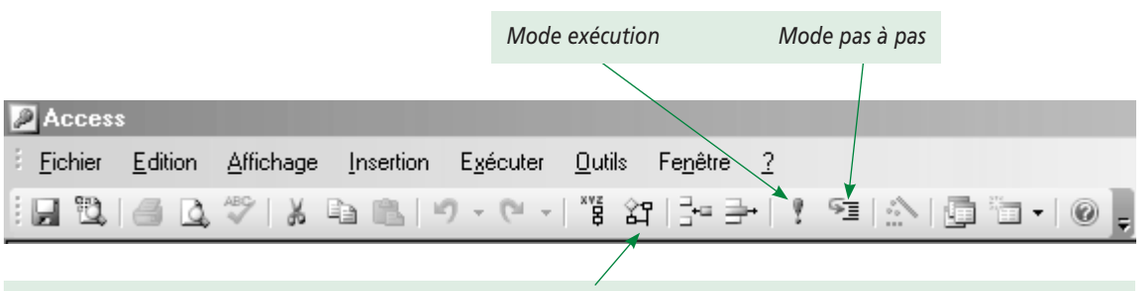
- **Comprendre une application Access utilisant des macros**

▶ *Contenu*

Synthèse

1. Introduction

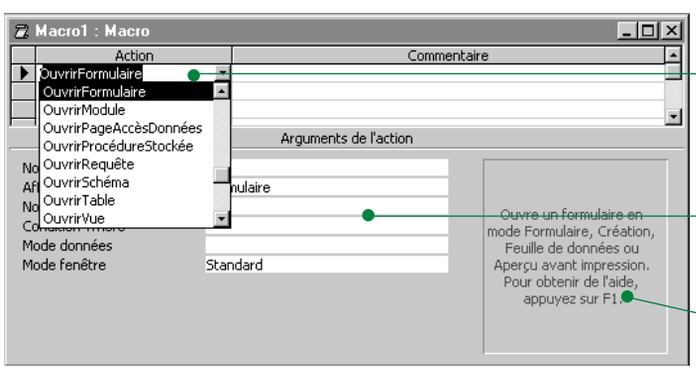
C'est la même notion que les macros *Word* ou *Excel* : elles permettent d'automatiser les tâches lourdes et répétitives. En effet, l'appel de la macro (un simple clic) lancera l'exécution automatique des actions programmées. On s'en sert pour gérer les passages d'un formulaire à un autre ou d'un menu à ses différentes commandes.
Depuis l'apparition du code VBA, les macros n'ont plus beaucoup d'intérêt. Elles sont conservées par souci de compatibilité ascendante et pour que ceux qui ne savent pas programmer puissent malgré tout automatiser un petit peu leur application.
En pratique, un sujet de stage classique peut consister à optimiser une ancienne base, éventuellement faite par un non-informaticien à l'aide d'assistants. Il vous faudra alors comprendre les macros… pour pouvoir les supprimer.
Un autre intérêt des macros : vous faites une maquette de l'application (c'est-à-dire les écrans et leur enchaînement sans aucun code) afin de montrer au client le principe de ce que vous voulez faire. Il est alors plus rapide d'utiliser les macros.
La barre d'outils *Création de macros* est la suivante. Les boutons les plus importants sont expliqués.



Affichage des conditions. Permet d'exécuter les actions de la macro selon des conditions (c'est le si *algorithmique).*

2. Création d'une macro

Pour créer une macro, sélectionnez les objets *Macros* depuis la fenêtre *Base de données* puis *Nouveau*. Vous accédez à la fenêtre de création. Une macro est constituée d'une suite d'actions qui sont exécutées l'une après l'autre dans l'ordre.



Dans la colonne Action, *vous avez la liste exhaustive des actions dans laquelle vous choisissez celle que vous voulez.*

Paramètres de l'action sélectionnée.

Aide sur l'action sélectionnée (testez !).

Pour sélectionner une action, il suffit de choisir celle qui vous convient dans la liste. Les arguments que vous devez fournir en-dessous indiqueront quels objets vous manipulez et comment. Par exemple, vous pouvez ouvrir un formulaire en filtrant ses enregistrements.

Une action est finalement une procédure qui s'ignore. L'action *OuvrirFormulaire* sera logiquement complétée par deux arguments définissant :

- le formulaire à ouvrir ;
- le mode d'ouverture du formulaire (réduit, agrandi, fenêtre modale…).

Pour en savoir plus, étudiez les diverses actions et leurs arguments.
Il sera possible de définir une macro ou un groupe de macros, c'est-à-dire qu'une macro peut avoir comme action l'exécution d'une autre macro (principe d'appel d'un sous-programme par un autre).
Pour exécuter une macro, vous pouvez cliquer le bouton *Ouvrir* de l'onglet *Macro* ou utiliser le bouton de raccourci.
En fait, une macro n'est généralement pas faite pour être exécutée à la main (en double-cliquant dessus). Elle permet d'automatiser les formulaires en étant déclenchée en réponse à un événement (voir… la suite).

3. Automatisation de formulaires

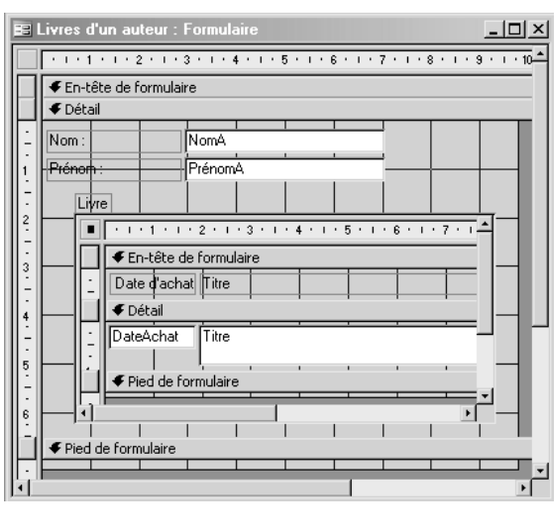
3A. *Présentation*

Pour automatiser le formulaire, la voie royale consiste à utiliser le code VBA. Mais on peut utiliser les macros :

- quand on ne sait pas programmer sous Access (ce qui est votre cas pour le moment) ;
- pour aller plus vite car la macro se rédige très rapidement.

Souvenez-vous des sous-formulaires vus dans la séquence 9 : le formulaire principal présente un enregistrement d'une table et le sous-formulaire présente les enregistrements liés d'une autre table. Dans le paragraphe 3B4, nous avons vu le formulaire principal présentant un auteur, le sous-formulaire listant les livres de l'auteur en question. Je vous rappelle le formulaire :

mode *Création*

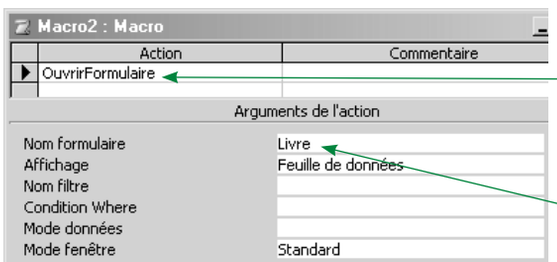


mode *Formulaire*

Il y a une alternative au sous-formulaire : mettre un bouton sur le formulaire principal, ce bouton ouvrant sur un clic souris un autre formulaire affichant les informations de l'ancien sous-formulaire. J'avais parlé du bouton dans la séquence 8 (paragraphe 2D). Nous allons maintenant nous en servir.

3B. Mise en œuvre : création de la macro

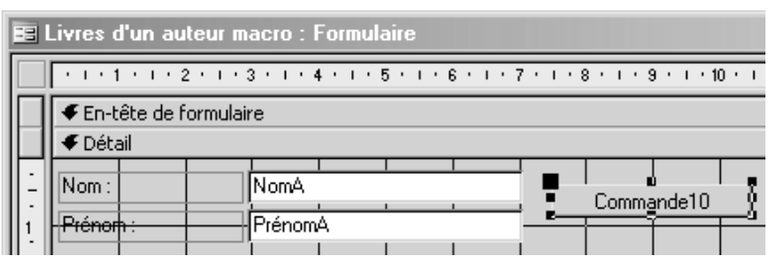
Nous allons le faire. Le formulaire principal contiendra le nom de l'auteur et un bouton affichant ses livres quand on clique dessus.
Dans un premier temps, il faut créer la macro permettant d'ouvrir le formulaire. À partir de la fenêtre *Base de données*, sélectionnez les objets *Macros* puis faites *Nouveau*. Créez la macro ainsi :



La macro n'est pas très longue... je n'ai qu'une action !

Nom du sous-formulaire. (Revoyez la troisième étape du paragraphe 3B4 de la séquence 9.)

Étudiez les différents paramètres de l'action. Sauvegardez-la ensuite sous le nom *Ouvrir*.
Je pars maintenant d'un formulaire basé sur la table *Auteur* et affichant le nom et le prénom. En prenant bien garde que le bouton *Assistants de contrôle* est **désactivé**❶, je place un contrôle *bouton de commande* dans le formulaire.



Je modifie ensuite trois propriétés du bouton :

- *Nom* (onglet *Autres*). Je mets *ListeLivres*. C'est plus parlant que le nom par défaut ;
- *Légende* (onglet *Format*). Je mets *Liste des livres* ;
- *Sur clic* (onglet *Événement*). Dans la liste déroulante de cette propriété, j'ai au choix *[Procédure événementielle]* et mes macros (enfin, vous ne devez avoir qu'une macro chez vous puisque c'est votre première). Choisissez la macro *Ouvrir*.

Les deux premières modifications ne doivent pas vous poser de problème. En revanche, il est sans doute utile de faire le point sur *Sur clic*. Nous sommes en pleine programmation événementielle. L'idée de la programmation événementielle est la suivante : les différents constituants d'une application (les formulaires, les contrôles…) exécutent du code en réponse à tous les événements qui les affectent. Par exemple, un contrôle bouton possède entre autres les événements suivants :

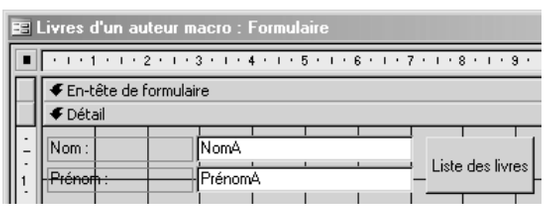
- *Sur clic* (cet événement se déclenche quand on clique sur le bouton) ;
- *Sur souris déplacée* (lorsque vous survolez le bouton avec la souris) ;
- *Sur double clic* (lorsque vous double-cliquez sur le bouton).

En observant la fenêtre *Propriétés* du bouton (onglet *Événement*), vous verrez que le bouton réagit à 12 événements différents. Si vous n'associez aucun code à un événement, il ne se passe rien lorsque ce dernier se produit. Dans notre cas, seul l'événement

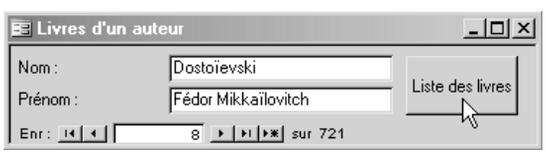
❶ *Sinon, l'assistant générera du code VBA, ce qui n'est pas notre objectif pour l'instant.*

Sur clic est renseigné. Ainsi, lorsque vous cliquerez sur le bouton, la macro *Ouvrir* s'exécutera. Lorsque tout autre événement se produira (survol de la souris, double clic…), il ne se passera rien puisque vous n'avez associé aucune action à ces événements. Notez que Windows lui-même a un fonctionnement événementiel (les deux événements principaux étant le clic et le double-clic). Ce que je viens d'expliquer est assez simple. Cela dit, chaque constituant d'une application Access (ou VB ou Delphi…) possède ses propres événements. Si *Sur clic* est trivial, d'autres sont beaucoup plus subtils❶.

Après avoir un peu redimensionné le bouton *ListeLivres*, on obtient le formulaire suivant :



À l'exécution :



En cliquant sur le bouton, j'obtiens :

Le formulaire s'ouvre bien en mode *Feuille de données* comme prévu. Seul problème… j'ai la liste de tous les livres et pas seulement ceux de l'auteur courant (on le voit dans la barre d'état : j'ai accès à l'intégralité des 2 321 livres).
Est-ce normal ? Oui, car ma macro demande l'ouverture du formulaire sans plus de précision. Je ne dis nulle part que je souhaite réaliser une jointure entre le formulaire présentant les auteurs et celui montrant les livres. Vous vous dites peut-être qu'Access pourrait bien s'en douter ? Et bien non, c'est impossible car un programme n'a pas accès à la sémantique.

Pour indiquer la jointure, il va falloir modifier la macro. Nous allons le voir ci-après.

❶ *À tel point que la compétence* développer avec un langage de programmation événementiel *doit être présentée dans l'épreuve de pratique informatique en option développeur d'applications.*

3C. Accès aux contrôles depuis la macro

3C1. Posons le problème

Le principe est simple. Dans la macro, on va dire :

- d'ouvrir le formulaire *Livre* (ce que l'on fait déjà d'ailleurs) ;
- d'afficher uniquement les enregistrements liés à celui du formulaire affichant les auteurs.

Comment exprimer cette dernière condition ? Si nous étions dans une requête, la condition de jointure entre les tables *Auteur* et *Livre* serait :
Auteur.NumAuteur = Livre.NumAuteur.
La différence est qu'ici les enregistrements à joindre ne viennent pas des tables *Auteur* et *Livre* mais des formulaires *Livres d'un auteur* et *Livre*.
Comment écrire notre condition de jointure ? Une solution consisterait à reprendre exactement la syntaxe des requêtes, ce qui donnerait :

Livres d'un auteur.NumAuteur = Livre.NumAuteur

Voyez-vous le problème ? Dans l'expression que je viens d'écrire, comment savoir ce que j'appelle *Livre* ? Fais-je référence à la table ou au formulaire ? En fait, si la question ne se pose pas avec *Livres d'un auteur*, c'est uniquement parce qu'il n'y a pas ambiguïté, aucune table ne s'appelant ainsi. La syntaxe SQL n'est donc pas adaptée. Il nous en faut une autre.

3C2. Résolvons le problème

Résumons-nous : nous devons indiquer que les enregistrements affichés par le formulaire *Livre* doivent avoir la même valeur de *NumAuteur* que celui affiché par le formulaire *Livres d'un auteur*.
Bref, il nous faut pouvoir accéder aux contrôles des formulaires. La syntaxe sera la suivante :

Formulaires!nom_du_formulaire!nom_du_contrôle

Il faut donc écrire le mot *Formulaires*, suivi du formulaire concerné et du contrôle visé, ces trois éléments étant séparés par un « ! ». Cela se lira : je veux le contrôle *nom_du_contrôle* du formulaire *nom_du_formulaire*. Si l'on sait qu'il s'agit d'un formulaire et non plus d'une table ou d'une requête, c'est parce que j'ai préfixé l'expression par *Formulaires*[1].

Lorsque je parle du nom du formulaire, il s'agit du nom identifiant, donc du nom sous lequel il est enregistré dans la base et pas de sa légende.

Par exemple, notre notation *Livres d'un auteur.NumAuteur* du paragraphe précédent devient :

Formulaires!Livres d'un auteur!NumAuteur

[1] *La lecture exacte est en fait : dans la collection des formulaires, je veux le formulaire XXX pour accéder au contrôle YYY. Mais bon, qu'est-ce qu'une collection ? C'est hors sujet ici (enfin la réponse est simple : c'est l'exact équivalent d'un ensemble mathématique). On retrouve cela sous Excel puisqu'un classeur est constitué d'une collection de feuilles. La différence principale entre une collection et un tableau ? Les éléments de la collection sont identifiés par un nom et sont en nombre variable.*

Une petite remarque

Vous observerez que *NumAuteur* **n'est pas** un contrôle du formulaire… Comment se fait-il alors que cela fonctionne ? Parce que, si ce n'est pas un contrôle, il fait néanmoins partie de la source de données qui alimente le formulaire. Donc, pour Access, il en fait partie. (Plus précisément, il est accessible à partir du formulaire.)

Je reprends ma note de bas de page pour vous donner un truc mnémotechnique : le mot *Formulaires* fait donc référence à l'ensemble (la collection) des formulaires. Il est au pluriel pour le distinguer du mot *Formulaire* qui est le type formulaire (cela signifiant d'ailleurs que vous pouvez sous VBA définir des variables formulaires). La collection des formulaires contient plusieurs formulaires… d'où le *s* !
Ouvrez la macro *Ouvrir* en modification, placez-vous sur la propriété *Condition Where* et lancez l'aide (*F1*). Lisez ce qu'Access vous explique. C'est exactement ce que nous voulons faire. Vous remarquerez que la syntaxe diffère sur deux points :

- j'utilise le mot *Formulaires* et dans l'aide, on parle de *Forms*. Ma foi, *formulaire* est la traduction anglaise de *Form*. En fait, tout le vocabulaire Access est dédoublé, en anglais et en français (car nous utilisons une version française). Vous pouvez donc taper le mot que vous voulez ;
- Access met les noms de formulaire et de contrôle entre crochets. Est-ce utile ? Je vous rappelle le rôle du crochet dans ce cours et celui de SQL : permettre d'identifier un nom contenant un espace (donc constitué de plusieurs mots). Les crochets ne seraient donc utiles pour nous qu'avec le nom *Livres d'un auteur macro*. Comme cette explication serait fastidieuse pour un non-informaticien, Access préfère mettre des crochets partout : c'est inutile mais cela fonctionne systématiquement, avec ou sans espace.

Ce qu'il faut retenir de l'aide, c'est que le *NumAuteur* du formulaire ouvert par la macro n'a pas à être préfixé par *Formulaires!Livre*. Pourquoi ? Car c'est implicite : si l'on ne fait référence à aucun formulaire, c'est que l'on parle du formulaire courant.
Souvenez-vous du paragraphe 2D de la séquence 9 : les expressions utilisaient les noms des contrôles sans préfixe car nous accédions à ceux du formulaire courant (celui qui est actif).

Notre condition *Where* sera donc :

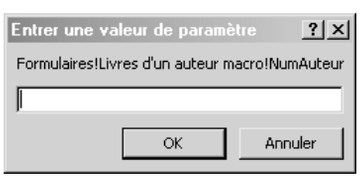
```
NumAuteur = Formulaires![Livres d'un auteur]!NumAuteur
```

En la tapant, vous remarquerez qu'Access ajoute des « [ ] » à tout le monde, ce qui donne finalement :

```
[NumAuteur]=[Formulaires]![Livres d'un auteur]![NumAuteur]
```

Vous retiendrez que le *NumAuteur* à gauche du « = » est implicitement celui du formulaire *Livre*. La lecture en français sera quelque chose du genre : « on n'affichera que les enregistrements de *Livre* dont la valeur du champ *NumAuteur* sera égale à la valeur de *NumAuteur* du formulaire *Livres d'un auteur* ».

Fermons tous les formulaires et exécutons la macro❶. On obtient la boîte de dialogue suivante :



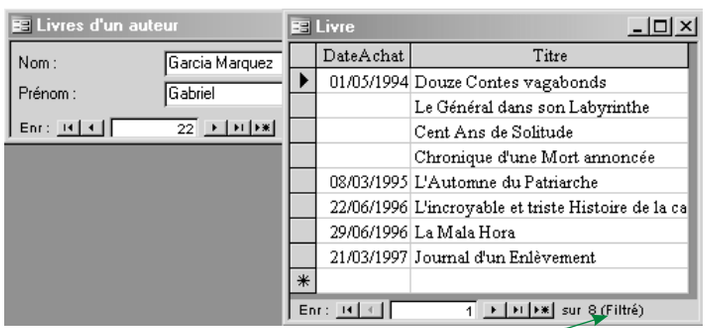
❶ *Commande* Exécuter/Exécuter *ou bouton* Exécuter *dans la barre d'outils. (Cela a déjà été vu.)*

Tout va bien ! Techniquement, la collection *Formulaires* contient les formulaires ouverts. Pour accéder à un formulaire à partir de cette collection, il faut donc qu'il soit ouvert. Cela est assez raisonnable : un formulaire fermé ne possède aucune valeur à afficher (aucun enregistrement courant dans sa source de données). Il est donc idiot de faire référence à son contenu.

La macro *Ouvrir* ne pourra donc fonctionner qu'avec le formulaire *Livres d'un auteur macro* ouvert. Cela tombe bien puisque cette macro doit être lancée par un bouton de ce formulaire. Nous retombons bien sur nos pieds.

3C3. Vérifions que cela fonctionne

J'affiche un auteur puis je clique sur le bouton et j'obtiens :



C'est parfait ! Vous remarquerez le mot *filtré* **ici**. Il signifie que les 8 livres affichés ne correspondent pas à l'intégralité du contenu de la table *Livre*, mais seulement à un sous-ensemble sélectionné (filtré) selon un critère précis.
Pour être complet, je vous précise que pour accéder à un contrôle dans un état, la syntaxe est similaire à celle employée pour les formulaires. Le principe est toujours de demander d'accéder à un état de la collection des états :

Etats!*nom_de_l'état!nom_du_contrôle*

(en anglais, on dira Reports *pour* États*).*

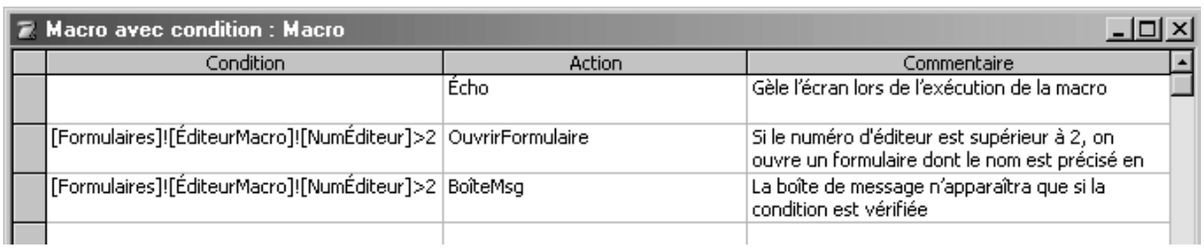
Pas question de mettre de majuscule accentuée : *Etats* est un mot réservé donc *États* ne serait pas reconnu.

4. Expressions conditionnelles

Il est possible d'exécuter ou non certaines actions de la macro selon qu'une expression booléenne est vérifiée. C'est la classique notion de test.

Les conditions s'écrivent dans une colonne qui n'est pas affichée par défaut. Pour la voir, menu *Affichage*/*Conditions* depuis la fenêtre d'édition de la macro (ou cliquez sur le bouton de raccourci correspondant).

Voici un exemple de macro utilisant une condition. Notez que chaque action mentionnée ci-dessous possède des propriétés que je ne détaille pas. Par exemple, pour geler l'affichage, il faut attribuer *Non* à la propriété *Écho actif* de l'action *Echo*. Je ne réactive pas l'affichage à la fin de la macro car Access le fait automatiquement❶.


Macro avec condition : Macro

Condition	Action	Commentaire
	Écho	Gèle l'écran lors de l'exécution de la macro
[Formulaires]![ÉditeurMacro]![NumÉditeur]>2	OuvrirFormulaire	Si le numéro d'éditeur est supérieur à 2, on ouvre un formulaire dont le nom est précisé en
[Formulaires]![ÉditeurMacro]![NumÉditeur]>2	BoîteMsg	La boîte de message n'apparaîtra que si la condition est vérifiée

La case *Condition* vide indique que l'action est toujours effectuée (sans condition). C'est le cas pour *Écho*.

La version algorithmique de la macro est :

Début
  Echo *non*
  si (*NuméroÉditeur* dans formulaire *ÉditeurMacro*) > 2 **alors**
    OuvrirFormulaire *paramètres*
    BoîteMessage *paramètres*
  fin-si
Fin

❶ *Comment sais-je tout cela ? Parce que j'ai consulté l'aide de l'action* Écho *pardi !*

Synthèse

Il faut retenir que la macro est une alternative au code VBA. C'est moins efficace (moins puissant et plus lent). Les seuls cas où vous devez les manipuler sont :

- lors de la maintenance d'une application ancienne ;
- pour tester la faisabilité d'un enchaînement de formulaires sans perdre de temps ;
- éventuellement pour réaliser simplement une toute petite automatisation.

Vous devez également retenir la façon d'accéder à un contrôle situé sur un autre formulaire ou état grâce à la syntaxe suivante :

- Formulaires!*nom_du_formulaire*!*nom_du_contrôle*
- Etats!*nom_de_l'état*!*nom_du_contrôle*

Ces deux notations seront utiles à chaque fois que l'on voudra, depuis un formulaire et/ou un état, faire référence au contenu d'un autre formulaire et/ou état (la seule contrainte évidente étant que ce dernier soit ouvert). On pourra donc s'en servir :

- dans les macros ;
- dans les expressions des propriétés *Source contrôle* des zones de texte ;
- dans le code VBA… que nous abordons dans la séquence suivante.

Séquence 12

Programmation VBA élémentaire

Après les macros, vestiges des temps anciens, il est temps d'aborder un outil moderne : le code VBA. Il ne s'agit pas d'un cours complet puisque VBA sera étudié en seconde année. Mon objectif est juste de vous montrer le principe de la programmation sous Access. Il ne s'agit donc pas de réaliser une application programmée complète, mais de savoir écrire des traitements simples automatisant une application.

▶ *Capacités attendues*

- **Pouvoir utiliser la génération automatique de code VBA par les assistants**
- **Savoir lire du code VBA et l'adapter légèrement**
- **Écrire du code VB simple (algorithmique) pour accéder aux propriétés des contrôles**

▶ *Contenu*

1. Introduction

Le langage VBA (*Visual Basic for Applications*, en français, cela donnerait « Visual Basic pour Applications ») est, comme son nom l'indique, une version particulière du langage VB.
En fait, c'est le langage Basic de VB auquel on rajoute les objets métiers de l'application support. Ainsi, le langage VBA :

- de Word gère les objets de Word (paragraphe, mot, phrase, tableau…);
- d'Excel gère les classeurs, feuilles, cellules, formules…;
- d'Access gère les tables, requêtes, formulaires, états, contrôles…

Le langage Basic est un langage procédural classique (comme le C, le Pascal…). Tous ces langages possèdent la notion :

- d'instruction (le test, la boucle…);
- de sous-programme (procédures et fonctions);
- de variable et de type (entiers, réels, chaîne…).

VBA possède tout cela en tant que langage Basic. Il possède aussi, comme je viens de le dire, l'accès aux objets métiers (en l'occurrence ici ceux d'Access). Comment faire pour accéder aux tables, requêtes, formulaires… depuis VBA ? Nous n'avons pas le choix : comme nous sommes dans le cadre de la programmation structurée, nous accéderons à ces objets à partir de variables.
Et, en programmation structurée, parler de variable revient à parler de type de donnée. VBA Access possède les types de données classiques (entier, chaîne…) mais aussi les types spécifiques formulaires, états, tables…❶
Nous avons déjà vu qu'Access et d'une façon générale VB manipule ces objets métiers sous la forme d'objets❷. Nous l'avions vu avec les contrôles, considérés comme des objets (des *choses*) ayant des caractéristiques (terme technique : propriétés) que l'on pouvait définir.
Ainsi, on trouvera deux types de variables sous VB ou VBA :

- les variables classiques (entiers, chaînes…) que vous manipulez depuis un certain temps déjà;
- toutes les autres (contrôles visuels, objets métiers) qui seront des objets.

Comme je l'ai dit dans ma note, VB n'est pas un langage objet. On peut donc voir les objets manipulés ici comme une simple sur-couche syntaxique à la notion classique de variable. D'ailleurs, la sémantique et la syntaxe des objets reprennent celles des types structurés.
J'essaie de vous faire comprendre qu'entre le formalisme (syntaxe) et la théorie des objets, il y a un monde que nous ne franchirons pas. Pour accéder aux concepts d'Access (tables, requêtes…), il nous faut les types de données correspondants. Comme ces objets sont volumineux (ils contiennent plus qu'une simple valeur) et datent de l'essor de la programmation objet, on utilise le formalisme objet pour les manipuler.
La programmation objet est étudiée en seconde année (option *Développeur d'applications* uniquement). Mais heureusement, nul besoin d'y connaître quoi que ce soit pour programmer sous VB(A). Tout ce qu'il faut savoir, c'est un peu de vocabulaire et de syntaxe que nous aborderons dans le paragraphe 3.

❶ *Bien entendu, les noms des types seront écrits en anglais !*

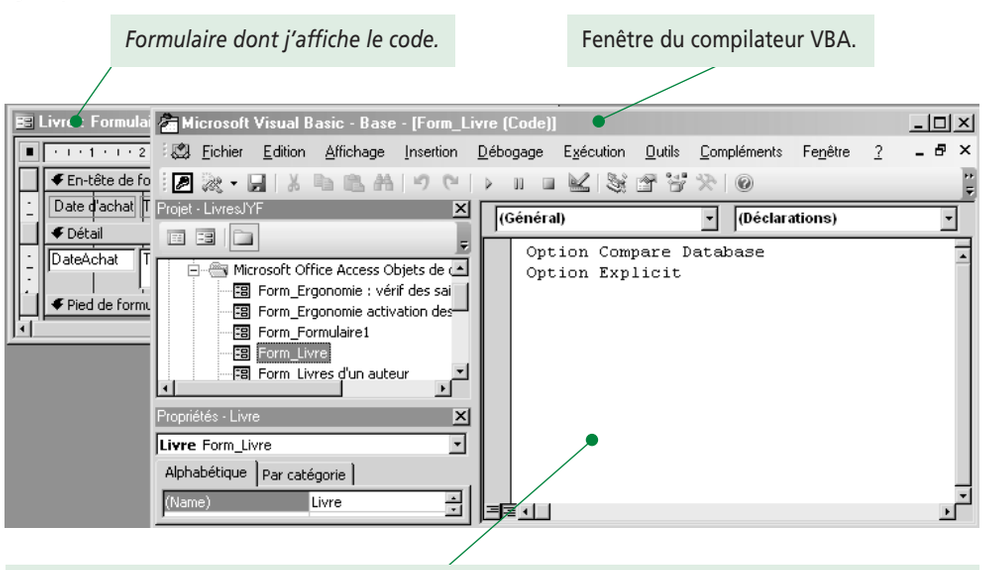
❷ *Attention, soyons précis : VB et VBA manipulent des objets mais ne sont pas pour autant des langages objets tels que C++, Java, Delphi ou VB.net car ils ne mettent pas en place tous les principes objets (héritage, encapsulation…). Ils n'en demeurent pas moins une approche parfaite du concept.*

Une remarque : dans Access 2003, vous trouverez alternativement VB (Visual Basic) ou VBA (Visual Basic pour Applications) selon les rubriques d'aides. Nous venons de voir que c'était plus ou moins la même chose. Disons que partout où vous voyez VB, remplacez mentalement par VBA. En fait, j'ai également trouvé dans l'aide le libellé « Microsoft Visual Basic Édition Applications ».

2. Le module

2A. Définition

Un module est un contenant dans lequel on met le code. On peut créer plusieurs modules par application, l'idée étant de pouvoir organiser son travail. Par exemple, on mettra tout le code VB relatif à un formulaire dans un module lié au formulaire.
Pour visualiser le code associé à un formulaire, ouvrez ce dernier puis menu *Affichage/Code* (ou bouton que je vous laisse chercher). Il existe un raccourci clavier caché : c'est *Alt+F11*.
S'il n'y a pas de code dans le formulaire, vous aurez une fenêtre vide :



Il est également possible d'ajouter des modules généraux accessibles à toute l'application. Notez pour votre édification que le type *Module* existe sous VBA. Cela signifie que vous pouvez créer du code de façon dynamique lors de l'exécution d'une application. Et cela, c'est neuf en programmation ! Votre code VB peut par exemple ajouter un bouton sur un formulaire et définir du code répondant à ses événements. Mais cela, c'est vraiment de la programmation avancée…

2B. Le code VBA

Le code VBA est le même que VB puisque les deux implantent le langage Basic. Ce que vous avez étudié dans le cours d'algorithmique est donc immédiatement applicable.
Je vous rappelle pour mémoire la syntaxe des différents éléments du langage.

2B1. Sous-programme

La syntaxe est la suivante :

FONCTION	PROCÉDURE
Public Function *nom_fonction* ()	**Public Sub** *nom_procédure* ()
…	…
nom_fonction = ❶	
End Function	**End Sub**

2B2. Programmation événementielle

VBA est un langage de programmation événementiel (pouvant répondre aux événements qu'il détecte). Les procédures pourront donc être appelées lors d'événements se déroulant dans des formulaires ou des états. Nous avons d'ailleurs vu dans la séquence précédente que l'on pouvait associer une macro ou du code VBA à l'événement *Sur clic* d'un bouton.

2B3. Les variables

Déclaration

Il existe deux modes de déclaration de variable : le mode explicite et le mode implicite. Le mode implicite vous évite de faire la moindre déclaration car le compilateur la fait lui-même à chaque nouvelle variable rencontrée dans le programme.
L'ennui, c'est que la moindre faute de frappe vous posera de gros problèmes : si vous utilisez la variable *NbrEmployés* et que, par une malencontreuse faute de frappe vous écriviez *NbrEmpolyés*, le compilateur va déclarer une nouvelle variable distincte de la première, ce qui va boguer tout votre code. Ainsi, il est beaucoup plus sain d'utiliser le mode explicite. Chaque variable utilisée doit alors être dûment déclarée. Les fautes de frappe seront donc détectées immédiatement.

Pour utiliser le mode explicite, il suffit d'ajouter *Option Explicit* dans la section *déclaration* du module (lignes au début du module, voir celui du paragraphe 2A). Si vous utilisez la déclaration explicite dans chaque module, le plus simple est de demander à VBA d'ajouter systématiquement cette ligne dans chaque nouveau module. Pour cela, allez dans *Outils/Options…*❷ onglet *Éditeur* (quand vous êtes dans un module, donc depuis la fenêtre VBA et non Access), cochez *Déclaration des variables obligatoire*.

Exercice

J'ai créé cet exercice pour que vous n'ayez aucune excuse si vous ne faites pas ce paramétrage. Activez la commande *Déclaration des variables obligatoire*.

Pour déclarer une variable, il faut utiliser le mot clé *Dim* ou *Global*. Il existe trois types de variables.

❶ *N'oubliez pas d'initialiser le résultat de la fonction !*

❷ *Attention ! Il s'agit d'un paramètre de VBA, pas d'Access. Il faut donc utiliser le menu* Outils *de la fenêtre VBA, pas celui de la fenêtre Access. Et pour cela, vous devez avoir ouvert la fenêtre VBA. (Rappel :* Alt+F11*, c'est rapide.)*

Les variables locales à un sous-programme

Elles ne sont utilisables que dans le sous-programme contenant leur déclaration (avec un *Dim*).

Les variables d'un module

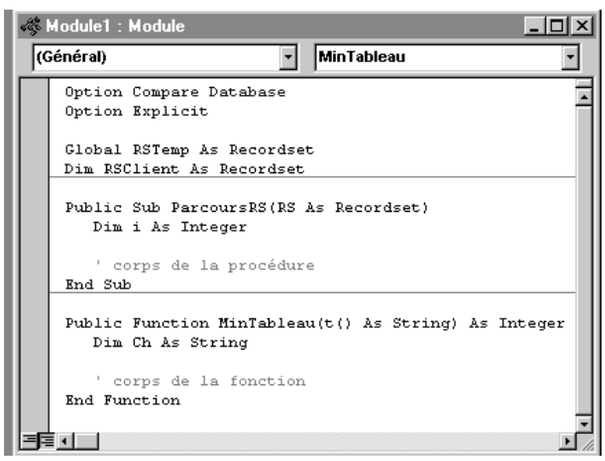
Elles sont déclarées avec un *Dim* dans la section de déclaration du module et sont accessibles à tous les sous-programmes du module.

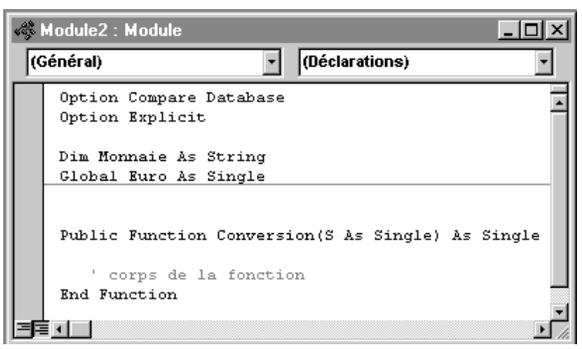
Les variables globales

Comme il faut bien les déclarer quelque part, elles le seront dans la section de déclaration d'un module. On utilisera le mot clé *Global* pour les distinguer des variables propres au module. Les variables globales sont accessibles dans tous les modules de l'application.

Exercice

Précisez, pour chaque sous-programme des deux modules suivants, les variables accessibles et inaccessibles. (La solution suit.)





Solution :

- *RSTemp* et *Euro* sont des variables globales donc accessibles partout, dans tous les modules ;
- *RSClient* et *Monnaie* ne sont accessibles que dans leurs modules de déclaration (*Module1* pour *RSClient* et *Module2* pour *Monnaie*) ;
- *i* et *Ch* sont des variables locales donc accessibles uniquement dans les sous-programmes qui les ont définies (*ParcoursRS* pour *i* et *MinTableau* pour *Ch*).

Types de variable

La déclaration d'une variable doit mentionner son type. Cela est fait avec le mot clé *as* suivi de l'un des types suivants (je ne présente que les plus importants) :

- *Integer :* nombre entier sur deux octets ;
- *Long :* nombre entier sur quatre octets ;
- *Single :* nombre réel sur quatre octets ;
- *Double :* nombre réel sur huit octets ;
- *String :* chaîne de caractères ;
- *Date :* date ;

En plus des types de base ci-dessus, il en existe une multitude d'autres propres à Access (*DataBase*, *RecordSet*…) ou non (*Currency*, *Boolean*…).

Notez que chaque variable doit être associée à son type (pas moyen de définir d'un coup 10 variables entières). Si vous déclarez une variable sans lui associer de type, Access lui donnera le type par défaut *Variant*. Cela revient à donner un type variable à votre variable. Celle-ci pourra donc contenir au fil du programme un entier, une chaîne, une date, une table… Cela semble sympathique (vous n'avez pas à définir le type et la variable peut contenir n'importe quoi, voilà un vrai progrès !) mais en fait, ce n'est pas efficace car Access doit convertir toutes les données et déterminer leur type. Votre code s'en trouvera extrêmement ralenti. Conclusion ? Et bien, n'utilisez **jamais** le type variant.

Exercice

Donnez le type des différentes variables déclarées ci-dessous :

```
Dim Compteur, i, NbrPlaces as Integer
Dim Nom as String
```

Solution :
Il y a une astuce, *Compteur* et *i* sont des variants car on ne leur associe aucun type. *NbrPlaces* est un entier, *Nom* une chaîne.

2B4. Les instructions

VBA est un langage de développement. Il contient donc les instructions de contrôle classiques.

Alternatives

```
if condition then
   instructions
end if
```

```
if condition then
   instructions
else
   instructions
end if
```

```
if condition then
   instructions
elseif condition then
        instructions
   else
        instructions
end if
```

```
select case expression
  case choix1
      instructions
  case choix2
      instructions
  .....
  case else
      instructions
end select
```

Boucles

La structure algorithmique *tant que… faire* se traduit de deux façons :

```
do while condition
  instructions
loop
```

```
do until not condition
  instructions
loop
```

La structure algorithmique *répéter… jusqu'à* se traduit de deux façons :

```
do
  instructions
loop until condition
```

```
do
  instructions
loop while not condition
```

La structure *pour* se traduit par :

```
for compteur = début to fin [step pas]
    instructions
next
```

3. Objets, propriétés et méthodes

Comme je vous l'ai dit dans l'introduction, vous devez avoir accès aux objets d'Access (tables, requêtes…). Pour cela, vous aurez les types de données correspondants puisque la manipulation de ces objets se fait avec une variable.

3A. Quelques objets métiers

Voici les principaux objets dont vous pouvez avoir besoin :

Objets d'Access	
Objet	**Type de donnée correspondant**
Base de données	DataBase
Requête (libellée en sql)	QueryDef
Formulaire	Form
État	Report
Table ou résultat d'une requête	RecordSet

Exemple :

```
Sub Exemple()
  Dim MonForm As Form, MonEtat As Report, Cpt As Integer

    Set MonForm = Forms![Saisie]
    Set MonEtat = Reports("Liste des employés")
    Cpt = 1
    //
    // instructions utilisant ces variables
    //
End Sub
```

Remarques sur cet exemple
Lorsque l'on manipule des objets Access, il faut toujours utiliser le mot clé *set* pour réaliser des affectations. *Set* est réservé aux objets et est interdit avec les variables de type classique (entiers, chaînes...) ; c'est là que l'on devine la technicité cachée des objets !
Les formulaires et états doivent être accédés par leur collection respective (*Forms* et *Reports*). Par exemple, le formulaire *Client* sera accédé au choix par :

- Forms![Client] // **notez bien le « ! »**
- Forms("Client")

Nous avions déjà vu la première formulation. Quel est l'intérêt de la seconde ? Le nom du formulaire est une chaîne de caractères (on le voit grâce à la présence des guillemets). On peut donc utiliser une variable, ce qui revient à paramétrer le formulaire.
Pour que le code ci-dessus fonctionne, le formulaire *Saisie* et l'état *Liste des employés* doivent être ouverts. La collection (l'ensemble) *Forms* contient les formulaires **ouverts** ; idem pour la collection des états *Reports*.

3B. Propriétés des objets

Nous avons déjà étudié les propriétés des objets visuels (états, formulaires, contrôles). Comme tout objet, ceux utilisés pour la programmation possèdent également des propriétés. En fait, tout objet visuel peut être accédé par programmation. Ainsi, dans le code ci-dessus, parler de la variable *MonForm* (une fois son initialisation faite) ou du formulaire *Saisie*, c'est pareil. *MonForm* a toutes les propriétés déjà vues dans les séquences 8 et 9.

Dans Access, tous les objets possèdent des propriétés qui peuvent être lues ou écrites lors de l'exécution du programme. Les propriétés des contrôles visuels peuvent être modifiées lors de la conception grâce à la fenêtre *Propriétés*. Nous l'avons vu lors de la création des formulaires. Toutes ces propriétés peuvent également être modifiées par programmation au fur et à mesure de l'exécution de l'application.

Par exemple, si la zone de texte *NuméroClient* est dans un formulaire *Saisie*, on peut écrire :

```
Forms("Saisie").NuméroClient.Visible = False
```

Que fait cette instruction ? Dans le formulaire *Saisie* (qui doit être ouvert), on accède au contrôle *NuméroClient* et on met sa propriété *Visible* à faux. Vous retrouvez l'opérateur « . » déjà observé avec les données structurées et permettant d'accéder à des variables (ou objets) imbriquées dans d'autres. Nous pourrions lire cette instruction de droite à gauche : on met à *faux* la propriété *Visible* du contrôle *NuméroClient* du formulaire *Saisie*. Concrètement, on aura l'impression que le contrôle disparaît lorsque cette instruction sera exécutée.

On pouvait écrire ce code en passant par une variable intermédiaire :

```
Dim f As Form
Set f = Forms("Saisie")
f.NuméroClient.Visible = False
f.NomClient.Visible = False
```

Qu'ai-je fait ? J'ai défini une variable intermédiaire *f* correspondant à mon formulaire. Cela me permet d'avoir un code plus simple lorsque j'accède à plusieurs contrôles (ou propriétés) de ce formulaire. Ici, je modifie les deux contrôles *NuméroClient* et *NomClient*.
Je vous rappelle que le but de cette séquence est de vous faire comprendre le code, pas de savoir en écrire de façon poussée. Cela, ce sera l'objectif de l'année prochaine.
L'utilisation classique de la programmation sous Access consiste à écrire du code attaché aux événements. Nous l'avons vu dans la séquence précédente puisque nous avions écrit une macro ouvrant un formulaire. Cette macro avait ensuite été attachée à l'événement *Sur clic* d'un bouton.

Eh bien, au lieu d'une macro, nous pouvons attacher du code VBA à un événement. L'intérêt vis-à-vis de la macro est évident :

- la macro est assez rigide et ne propose qu'un nombre restreint d'actions. Vous pouvez faire pas mal de choses (ouvrir ou fermer un formulaire, changer d'enregistrement courant…), mais pas tout ;
- le code VBA est un méta-langage d'Access, ce qui signifie que vous pouvez écrire Access en VBA. Ainsi, tout Access vous est accessible au travers de VBA. Cela va très loin : non seulement vous avez la maîtrise totale des objets de la base (formulaires, tables, requêtes…) mais vous pouvez rajouter ce que vous voulez. Vous pouvez par exemple rajouter dynamiquement (en cours d'exécution de l'application) des tables, des formulaires mais aussi des relations, des modules et des contrôles sur des formulaires.

Au vu de cette comparaison, il est évident que le langage VBA doit être votre interlocuteur privilégié sous Access. Il faut cependant admettre que la maîtrise de VBA passe non seulement par la connaissance de VB, mais aussi par la compréhension des différents objets Access. Il y a donc un investissement (en travail) sans commune mesure avec celui nécessaire à l'utilisation des macros.
En pratique, développer une base de données réelle sans programmation est illusoire. Vous apprendrez la programmation Access en seconde année. Cette séquence a juste pour objet de vous faire comprendre l'importance de cette programmation et vous donner les connaissances suffisantes pour lire et comprendre un programme VBA, ce qui est une étape préalable à savoir en écrire. Nous nous contenterons d'écrire du code VB simple sans manipuler les objets Access.

3C. Propriétés et méthodes des objets

3C1. Présentation

Il ne s'agit pas ici de vous faire un cours de programmation objet, mais juste de vous expliquer la syntaxe à venir.
En mode *Création* des formulaires, vous aviez accès aux propriétés par l'intermédiaire de la fenêtre *Propriétés*. Je vous rappelle que les propriétés sont les caractéristiques de l'objet :

- une voiture est caractérisée par sa couleur, sa marque, sa cylindrée, son modèle, son nombre de places… ;
- un formulaire aura notamment les propriétés *couleur*, *source de données*…

Un objet n'est pas limité à ses propriétés. Il peut également réaliser des traitements. Par exemple :

- l'objet *voiture* peut accélérer, ralentir, démarrer, s'arrêter, débrayer… ;
- un formulaire peut s'afficher, se dessiner… ;
- un tableau peut se trier, rechercher un élément, s'ajouter ou se supprimer un élément, s'afficher… ;
- un module (du code VBA) peut s'ajouter ou se supprimer du code, il peut rechercher du texte dans son code…

Le terme technique dans le vocabulaire objet n'est pas *traitement* mais *méthode*. Ainsi, on dira que l'objet *voiture* a :

- pour propriétés *couleur*, *marque*, *modèle*… ;
- pour méthodes *accélérer*, *démarrer*, *ralentir*…

Ce qu'il faut retenir, c'est qu'un objet :

- est caractérisé par ses propriétés qui le distinguent d'un autre objet du même type. Par exemple, ce qui distingue deux voitures, ce sont bien leur marque, couleur, cylindrée… ;
- peut réaliser des traitements (des actions) sur lui-même grâce à ses méthodes.

C'est une analogie avec la vie réelle : chaque objet qui nous entoure possède des caractéristiques qui l'identifient et peut réaliser certaines actions.
Une question intéressante est : d'où viennent les propriétés et les méthodes ? En fait, elles ne sont absolument pas arbitraires. Pour les identifier, il faut réfléchir (on retombe dans l'analyse). Vous voulez gérer un objet *Compte bancaire*. C'est à vous de découvrir que cet objet aura :

- comme propriétés *numéro*, *date d'ouverture*, *solde* et *titulaire* ;
- comme méthodes *ouverture*, *clôture*, *créditer*, *débiter*.

Il est évident que la propriété *couleur* et la méthode *accélérer* n'ont aucun sens pour cet objet. Cela dit, il en existe certainement d'autres (je réfléchis… oui, un compte est caractérisé par sa devise : dollar, euro…). Les différentes caractéristiques ne tombent donc pas du ciel. Ainsi :

- quand vous développez une application objet, vous devez réfléchir de façon abstraite (avec une méthode d'analyse) sur les caractéristiques des objets (propriétés et méthodes) ;
- quand vous utilisez des objets (par exemple les objets formulaires, contrôles…) en développant, vous devez étudier les différentes propriétés et méthodes que le concepteur de l'objet met à votre disposition. Et pour cela, que ce soit vous ou moi… il faut utiliser l'aide ! Pas moyen de deviner tout cela.

3C2. Formalisme

Notez que d'un point de vue technique, les méthodes sont des sous-programmes (soit procédures soit fonctions). Cela vient du fait que l'on est toujours dans le cadre de la programmation structurée.
Dans le paragraphe précédent, je n'ai pas parlé des éventuels paramètres associés aux méthodes. Mais bon, cela vient naturellement : lorsque j'appelle la méthode *accélérer* de la voiture, il est logique d'indiquer la vitesse à atteindre. De même, lorsque l'on crédite un compte bancaire, il faut indiquer de quel montant on le crédite !
Il faut donc bien faire la différence entre les concepts (objet, propriétés et méthodes) et la façon dont on les implante (variables et sous-programmes). Pour accéder aux méthodes ou aux propriétés d'un objet, on utilise le désormais classique opérateur « . ».
Le formalisme général est le suivant :

```
variable = objet.méthode (arguments)     si la méthode retourne un résultat (= fonction)
objet.méthode (arguments)                si la méthode ne retourne aucun résultat (= procédure)
variable = objet.propriété               lecture d'une propriété
objet.propriété = valeur                 écriture (modification) d'une propriété
```

Par exemple, si j'ai un compte bancaire *MonCompte* (variable de type *Compte*❶) et que je fais un dépôt de 500 €, j'écrirai :

```
MonCompte.Créditer (500)
```

De même, si je veux clôturer le compte *MonVieuxCompte* et transférer l'argent dans le compte *MonNouveauCompte* que je crée pour l'occasion, j'aurai :

```
[1]  si MonVieuxCompte.solde < 0
[2]  alors
[3]     afficher ("on ne peut clôturer un compte débiteur")
[4]  sinon
[5]     MonNouveauCompte.Ouverture
[6]     MonNouveauCompte.Créditer (MonVieuxCompte.Solde)
[7]     MonVieuxCompte.Débiter (MonVieuxCompte.Solde)
[8]     MonVieuxCompte.Clôturer
[9]  fin-si
```

Cet exemple nécessite quelques explications :

- le principe est le suivant : je crée mon nouveau compte, je le crédite du solde de l'ancien compte, je vide l'ancien compte puis je le clôture ;
- ligne [5] : je suppose visiblement que la méthode *Ouverture* renseigne les diverses propriétés (le numéro de compte est généré, le solde est mis à 0 et la date d'ouverture est la date du jour) ;
- la ligne [7] revient à débiter le compte du montant exact de son solde. Je pouvais tout aussi bien écrire : *MonVieuxCompte.Solde = 0* ;
- cet exemple est utile pour la syntaxe des objets. Il n'est pas très réaliste quant à l'informatique bancaire : si le programme plante (erreur, coupure de courant) entre les lignes 6 et 7, je deviens l'heureux possesseur de deux comptes approvisionnés !

❶ *Soyons précis : le vocabulaire objet dira que* ***l'objet*** MonCompte *est une* ***instance*** *de la* ***classe*** Compte.

Je n'ai donné aucun exemple de méthode qui soit une fonction. En fait, en programmation objet, on évite d'accéder directement aux propriétés. On passera donc plutôt par des méthodes. C'est pour cela que j'ai créé *créditer* et *débiter*. Ces méthodes ne sont pas utiles d'un point de vue syntaxique puisque :

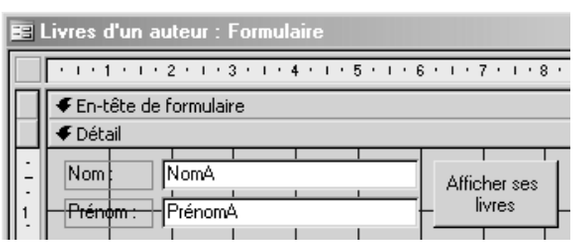
Compte.créditer (1000) revient à Compte.solde = Compte.solde + 1000

On peut donc créer une méthode *DonneSolde* qui se contente de renvoyer la valeur de la propriété *solde*.
Comme je l'ai déjà dit, VB (et VBA) ne sont pas des langages objets, même s'ils manipulent des variables qui sont des objets. On s'autorise donc à accéder aux propriétés quand on en a besoin.

4. 1er exemple : boutons de navigation programmés

4A. Utilisons l'assistant

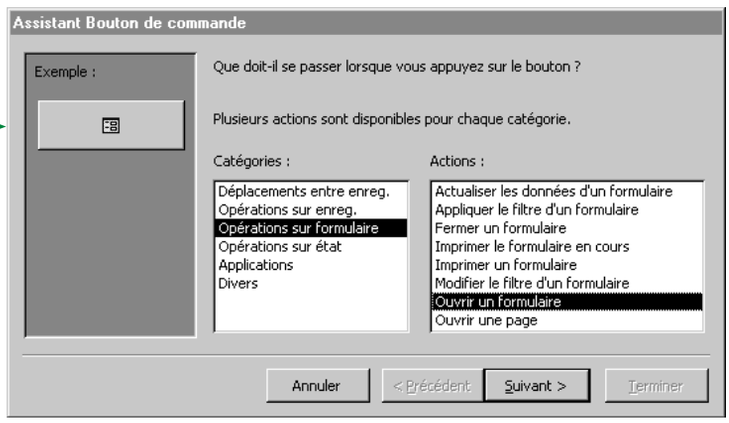
Au risque de vous ennuyer, je me répète : ce cours a pour objet de vous permettre de **lire** du code VBA et **d'écrire** du code VB simple (algorithmique).
Nous allons utiliser l'assistant du contrôle *Bouton de commande*. Reprenez le formulaire établi dans la séquence précédente :



Supprimez le bouton (sélectionnez-le puis touche *Suppr*). Assurez-vous que le bouton *Assistants de contrôle* est appuyé dans la boîte à outils puis placez un bouton de commande❶ dans le formulaire.
Cela va déclencher un assistant vous demandant ce que vous voulez faire avec ce bouton puis quel dessin ou texte le bouton doit contenir. Le principe est simple : l'action que vous allez choisir va être codée en VBA et ce code sera associé à l'événement *Sur clic* du bouton.
La première fenêtre de l'assistant vous propose les différentes actions possibles. Notez bien que l'on retrouve les actions des macros. Si vous voulez faire un traitement qui n'est pas dans la liste, c'est tout à fait possible, mais plus question d'utiliser l'assistant : il faut écrire le code soi-même.

❶ *Attention à l'erreur bête : prenez un* bouton de commande*, pas un* bouton bascule.

Le dessin proposé est celui habituellement utilisé pour l'action choisie.

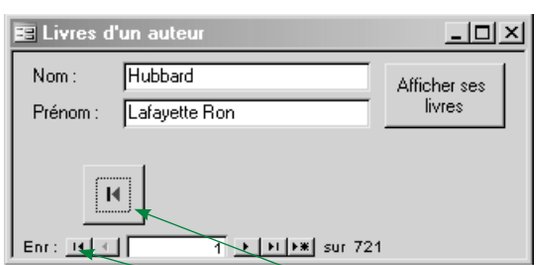


Étudiez les différentes catégories et les actions correspondantes. Choisissez ensuite la catégorie *Déplacements entre enreg.* et l'action *Premier enregistrement*.
La fenêtre suivante vous permet de choisir l'illustration du bouton :

- du texte à rentrer ;
- une image parmi celles proposées ;
- n'importe quelle image *bitmap* sur votre ordinateur.

Je vous conseille de conserver l'image proposée par défaut car elle correspond au dessin standard vis-à-vis de l'action. L'utilisateur s'y retrouvera plus facilement.
Enfin, vous devrez comme d'habitude donner un nom au bouton (pour faire le lien avec ce qui précède, ce nom sera utilisé pour renseigner la propriété *Nom* du contrôle). Je vous propose *PremierEnreg*.
Passez en mode formulaire et testez le bouton : affichez le 5e auteur grâce à la barre d'état puis appuyez sur votre nouveau bouton.

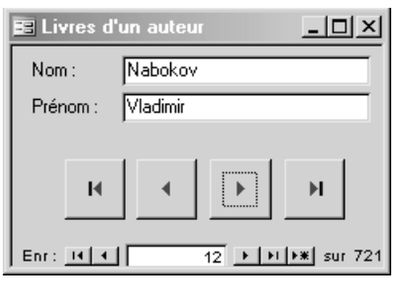
En cliquant sur le bouton, vous obtenez le formulaire ci-contre.



*En fait, les deux boutons (**celui-ci** et **celui-là**) ont un comportement identique.*

Pourquoi s'amuser à refaire un bouton qui existe déjà ? Tout simplement car la barre d'état est toute petite et pas très lisible. De plus, elle propose des informations (enregistrement courant, nombre d'enregistrements) et une action (ajout d'un nouvel enregistrement) que l'on ne souhaite pas forcément proposer.
Ainsi, pour des raisons d'ergonomie, il est préférable de refaire ces boutons. Vous remarquerez que l'assistant *Bouton* propose l'action *ajouter un nouvel enregistrement* (dans la catégorie *Opérations sur enreg.*). Je ne m'en servirai pas puisque je ne veux que me déplacer dans les enregistrements.
Placez trois nouveaux boutons réalisant les actions *Enregistrement précédent*, *Enregistrement suivant* et *Dernier enregistrement*. Vous les appellerez respectivement *EnregPréced*, *EnregSuivant* et *DernierEnreg*.

Déplacez s'il le faut les boutons pour obtenir le formulaire suivant :



Testez les boutons… tout doit bien se passer. Maintenant, est-il utile de conserver la barre d'état du formulaire ? Non, puisque les fonctions utiles (déplacement dans les enregistrements) viennent d'être reprises.
Enlevez-la (si vous avez oublié la technique, allez séquence 7, paragraphe 3D2). On obtient :

Allons maintenant voir le code VBA qui a été généré (menu *Affichage/Code* lorsque le formulaire est en mode *Création* ou le bouton associé ou *Alt+F11*).
Vous aurez plusieurs procédures, dont l'une est *PremierEnreg_Click()*.Vu son nom, on se doute qu'elle est exécutée lorsque l'événement *Sur clic* est déclenché sur le bouton *PremierEnreg*. Le nom des procédures événementielles sera toujours constitué du nom du contrôle suivi d'un tiret et du nom de l'événement (*Sur clic* se dit *Click* en anglais).

Notez que seule l'instruction *DoCmd.GoToRecord , , acFirst* est intéressante pour nous. Les autres lignes permettent de gérer les erreurs d'exécution en exécutant une partie de code précise. Ignorez-les.

Nous allons maintenant refaire le bouton permettant d'afficher les livres d'un auteur donné. Ce sera plus simple qu'avec la macro car l'assistant nous demandera le lien entre les différents enregistrements.

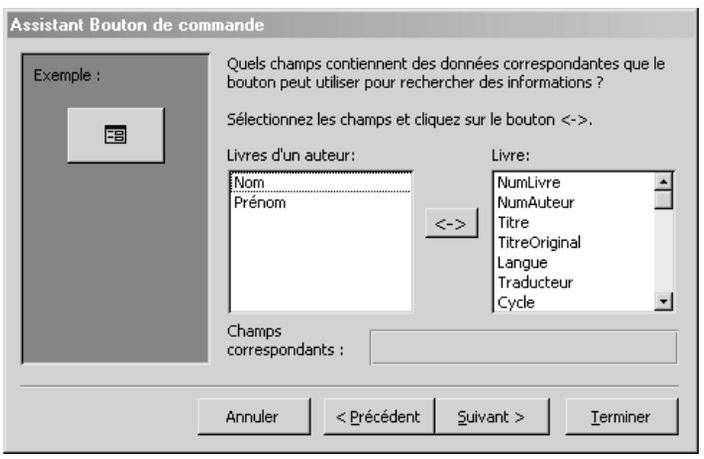
Affichez le formulaire en mode *Création* et rajoutez un bouton. Utilisez l'action *Ouvrir un formulaire* (je vous laisse le soin de trouver la catégorie correspondante).

L'assistant vous demande quel formulaire vous voulez ouvrir. Choisissez celui utilisé dans la macro (séquence 11, paragraphe 3B).

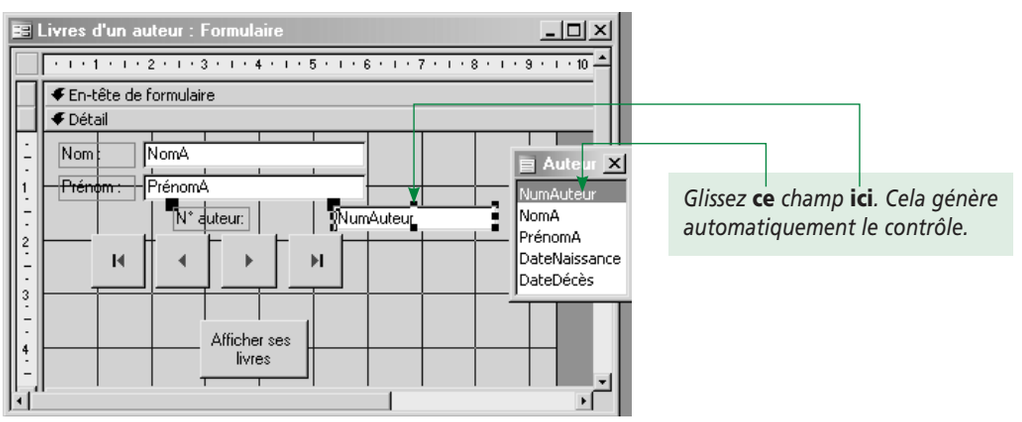
L'assistant vous demande ensuite si vous voulez afficher tous les enregistrements du formulaire ou un sous-ensemble spécifique. Comme nous ne voulons afficher que les livres de l'auteur courant, optez pour la première proposition (*Ouvrir le formulaire et trouver des informations spécifiques à afficher*).

L'écran suivant vous demande de façon plus ou moins claire le critère pour afficher ou non un enregistrement du formulaire en cours d'ouverture. En pratique, il faut sélectionner la clé primaire d'un côté, la clé étrangère de l'autre, puis cliquer sur le bouton central.

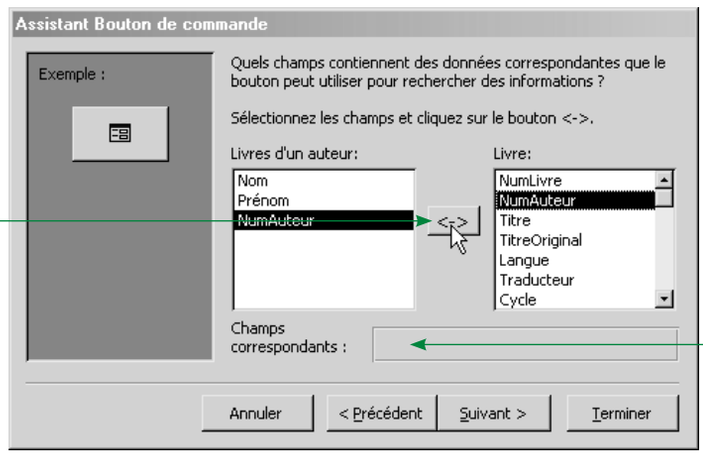
Voici la fenêtre :



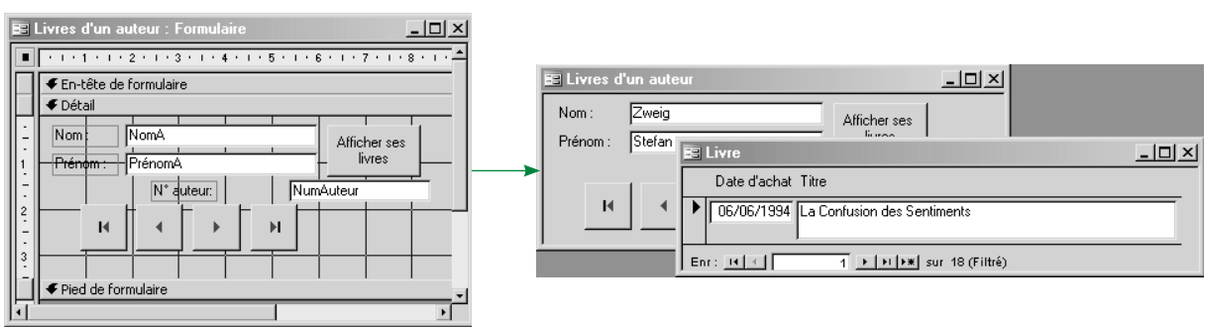
Là, nous avons un gros problème : l'assistant propose uniquement les champs affichés dans le formulaire principal et non les champs de sa source. *NumAuteur* n'apparaît donc pas. Que faire ? On va annuler l'assistant (cliquez sur *Annuler*), supprimer ensuite le bouton créé puis, à partir de la fenêtre *Liste des champs*, glisser *NumAuteur* dans le formulaire (revoyez la séquence 6 si cela ne vous semble pas clair).



Comme on ne veut pas pour autant afficher ce champ, mettez sa propriété *Visible* à *Non*. Comme il ne sera pas affiché, je le laisse *en vrac* dans le formulaire.
Relancez l'assistant jusqu'à obtenir la fenêtre qui nous avait bloqués. Cette fois, on peut choisir nos champs comme le prouve l'illustration suivante.



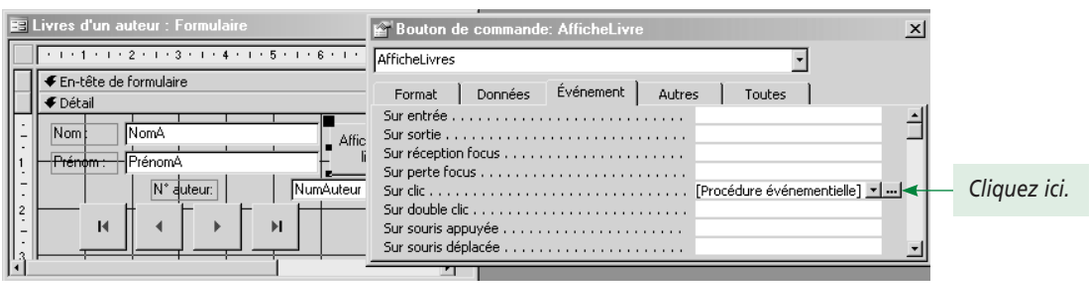
Une fois **ce** bouton cliqué, la correspondance entre les deux champs s'affichera **ici**.
La fenêtre suivante vous demande l'illustration à mettre sur le bouton. Je vous propose le texte *Afficher ses livres*. Enfin, le nom du bouton sera *AfficheLivres*. Vérifions que cela fonctionne :



Cela fonctionne pas mal… Le seul ennui, c'est que le formulaire *Livre* est visiblement ouvert en mode *Formulaire unique* et non *Feuille de données* : on ne voit qu'un livre à la fois alors que nous les voudrions tous d'un coup.
Comme l'assistant ne nous a pas demandé le mode d'ouverture et que celui utilisé ne convient pas, il nous faut aller modifier le code qui ouvre le formulaire.

4B. *Optimisons le code de l'assistant*

Pour afficher le code déclenché lors de l'événement *Sur Clic* du bouton, le plus simple est de sélectionner ce dernier en mode *Création* du formulaire puis de cliquer sur le bouton … au bout de la ligne de l'événement *Sur Clic* de la fenêtre *Propriétés* (illustration ci-dessous !) :

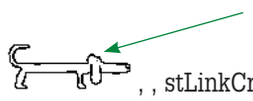


Vous arrivez alors directement sur la procédure *AfficheLivres_Click*. Si l'on fait abstraction du code gérant les erreurs, l'instruction ouvrant le formulaire est :

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

C'est assez intuitif : *DoCmd.OpenForm* signifie *Exécute la commande* « *Ouvrir Formulaire* ». Il nous faut plus d'informations sur cette instruction pour savoir comment ouvrir le formulaire dans le mode que l'on souhaite. Cliquez sur le mot *OpenForm*, puis *F1* pour avoir l'aide.
Tout s'explique : on vous dit que le second paramètre *Afficher* (qui est facultatif d'après l'aide) détermine le mode d'affichage. Lisez toute l'aide sur cet argument. Vous noterez que si l'argument n'est pas renseigné le formulaire est ouvert en mode normal *acNormal* (un enregistrement à la fois).
Et c'est bien notre cas, car, s'il était présent, il devrait se trouver **là** où j'ai mis un teckel :

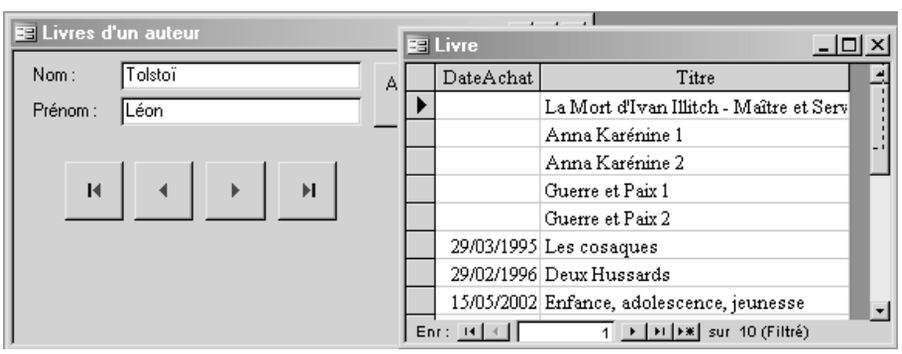
```
DoCmd.OpenForm stDocName,  , , stLinkCriteria
```



Il suffit donc de trouver le bon mode d'affichage. Pas de miracle : le plus simple est de les tester un par un jusqu'à trouver le bon. On découvrira que le mode *Feuille de données* correspond à *acFormDS*.
Modifiez donc l'instruction ainsi (attention à ne pas vous tromper dans les virgules) :

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

Sauvegardez puis revenez au formulaire et testez :



C'est parfait !
Allons encore plus loin : l'assistant a généré du code que nous n'aurions pas inventé (en fait, le *DoCmd.OpenForm* indique qu'il a utilisé un objet *DoCmd* dont il a appelé la méthode *OpenForm*).

Le problème, c'est que :

1. Le code généré n'est ni optimisé ni lisible. C'est le cas de toute génération automatique de code. C'est très brutal et assez difficile à exploiter. Créez des macros sous Word ou Excel puis allez voir le code VBA généré… c'est impressionnant !
2. Tout n'est pas forcément utile. Par exemple, la gestion d'erreur est inutile ici car la seule erreur qui puisse arriver est la disparition du formulaire… que mon application plante serait alors le cadet de mes soucis.

Nous allons modifier le code généré. Dans un premier temps, supprimons tout ce qui concerne la gestion d'erreur.

Il restera :

```
Private Sub AfficheLivres_Click()

  Dim stDocName As String
  Dim stLinkCriteria As String

  stDocName = "Livre"
  stLinkCriteria = "[NumAuteur]=" & Me![NumAuteur]
  DoCmd.OpenForm stDocName, acFormDS, , stLinkCriteria
End Sub
```

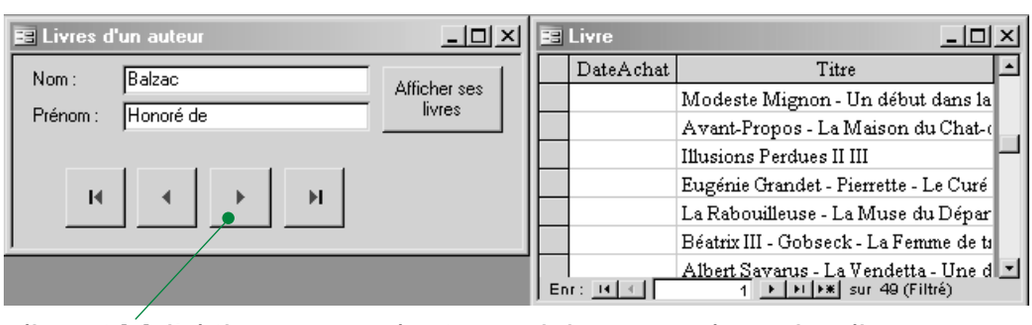
Enfin, on peut aisément se passer des deux variables intermédiaires *stDocName* et *stLinkCriteria*. On les remplace donc par leur valeur, ce qui donne :

```
Private Sub AfficheLivres_Click()
  DoCmd.OpenForm "Livre", acFormDS, , "[NumAuteur]=" & Me![NumAuteur]
End Sub
```

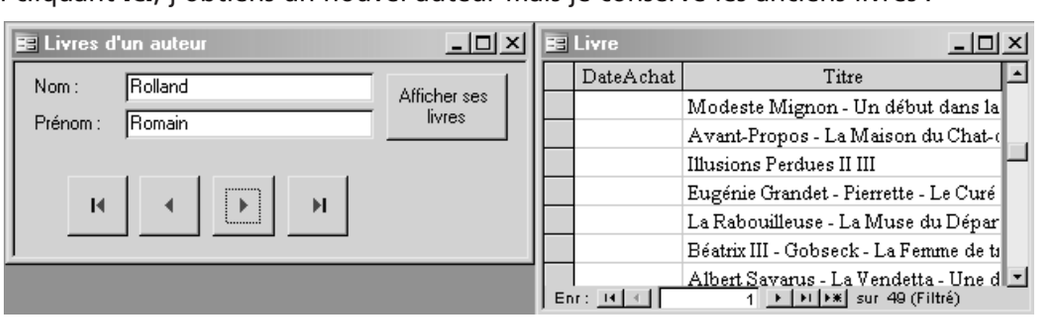
Voilà ! On a singulièrement simplifié le code ! Nous verrons dans la séquence suivante que certaines gestions d'erreurs doivent impérativement être conservées car elles gèrent des erreurs non fatales qui sont plus des erreurs de manipulation de la part de l'utilisateur que des bugs. Par exemple, l'accès à un enregistrement inexistant.

4C. Complétons modestement le code généré

Affichez un auteur quelconque, cliquez pour obtenir ses livres puis changez d'auteur. Que se passe-t-il ? Rien : les livres sont toujours ceux de l'ancien auteur.
Voici les livres de Balzac.



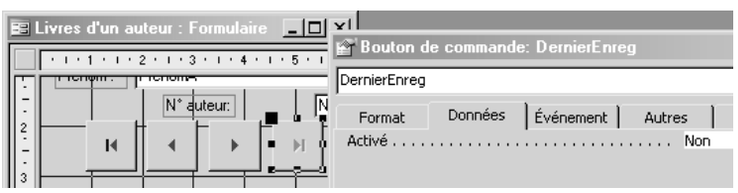
En cliquant **ici**, j'obtiens un nouvel auteur mais je conserve les anciens livres :



Comment régler ce problème ? Une solution consiste à dire que dès que l'on clique sur un bouton de déplacement alors que la fenêtre des livres est ouverte, on ferme cette fenêtre et on la réouvre avec les bons livres. Cette solution est un peu complexe pour nous.

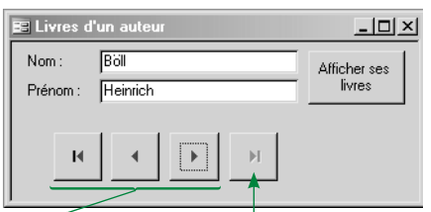
Nous allons faire plus simple en interdisant que cette situation puisse se produire. L'idée sera de faire ce qu'il faut pour que l'affichage des livres rende les boutons de déplacement inactifs : cliquer dessus ne produira rien.
Comment faire cela ? On pourrait supprimer le code VBA associé au bouton… difficile à faire par programmation et assez violent. Il y a beaucoup plus simple : utiliser la propriété *Activé* des boutons. Testez cela à la main : dans le formulaire affiché en mode *Création*, mettez la propriété *Activé* du bouton *DernierEnreg* à *Non*. Observez que le bouton devient gravé en creux et sans couleur. C'est l'aspect normalisé sous Windows pour les commandes ou les boutons inactifs. Vérifiez qu'en passant au mode *Formulaire*, cliquer sur le bouton ne fait plus rien.

mode *Création*



Ces trois boutons demeurent actifs.

mode *Formulaire*



Je clique, je clique… il ne se passe rien !

(Une fois le test fait, réactivez le bouton.)

Bien ; nous savons comment activer ou non les boutons… mais quand le faire ? Parce qu'ici, liste des livres affichée ou non, je n'ai plus accès à mon bouton *DernierEnreg*.
C'est un problème de programmation événementielle. Je vous propose de désactiver les boutons de déplacement lorsque l'on clique sur le bouton *Afficher ses livres*, juste avant l'ouverture du formulaire. Nous n'utiliserons donc pas la fenêtre *Propriétés*, mais le code VBA. Comment l'écrire ? Relisez si nécessaire le paragraphe 3C2. La syntaxe pour modifier la valeur de la propriété d'un objet est :

objet.propriété = valeur

On veut donc écrire DernierEnreg.Activé = non. Le problème, c'est que si les propriétés sont en français (c'est plus accessible que l'anglais à l'utilisateur moyen), le code VBA est réservé aux informaticiens. Et là, l'anglais est de rigueur. Comment traduire *Activé* ? Dit autrement, quelle propriété en anglais utiliser ? Placez-vous dans la fenêtre *Propriétés* sur la propriété *Activé*, lancez l'aide par *F1*… le début de l'aide vous indique❶ :

Propriétés Enabled (Activé), Locked (Verrouillé)

Voir aussi Exemple S'applique à

- La propriété **Activé (Enabled)** spécifie si un contrôle peut être actif en mode Formulaire.

C'est réglé… *Activé* se dira *Enabled*. N'hésitez pas à cliquer sur *Exemple* pour voir des exemples d'instructions VBA utilisant cette propriété. Cela vous indiquera d'ailleurs que *Non* se dira *False*.

Notre instruction est alors :

DernierEnreg.Ena bled = False (et idem pour les autres boutons).

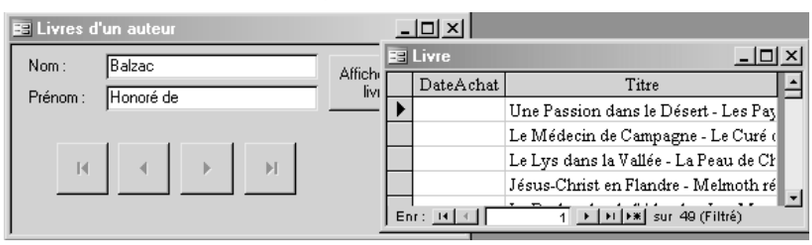
❶ *Je triche : la copie d'écran que je vous propose vient de l'aide d'Access 2000. Vous aurez donc un autre texte. Je conserve ma version car elle est plus explicite au vu de ce que je souhaite expliquer. Évidemment, la propriété* Activé *possède la même signification dans les deux versions.*

Retournez dans la fenêtre de code et modifiez la procédure *AfficheLivres_click* ainsi (essayez de le faire vous-même avant de lire la solution qui suit) :

```
Private Sub AfficheLivres_Click()
  PremierEnreg.Enabled = False
  EnregSuivant.Enabled = False
  EnregPréced.Enabled = False
  DernierEnreg.Activé = False
  DoCmd.OpenForm "Livre", acFormDS, , "[NumAuteur]=" & Me![NumAuteur]
End Sub
```

Testez ! Tout va presque bien : je choisis un auteur…

J'affiche ses livres : les boutons de déplacement sont désactivés.



Le problème, c'est que lorsque je ferme la fenêtre des livres…

Les boutons de déplacement restent inactifs !
Bah oui, je n'ai écrit nulle part qu'il fallait réactiver les boutons… Il ne nous reste plus qu'à le faire. Comment le faire, c'est trivial (j'affecterai *True* aux propriétés *Enabled*). Mais où le faire ?

Encore une fois, c'est un problème de programmation événementielle. Quand veux-je réactiver mes boutons ? Au choix :

- lorsque je ferme le formulaire *Livre* ;
- quand je reviens au formulaire *Livres d'un auteur*.

Quels événements sont concernés ? Ma foi, *Sur fermeture* dans le premier cas et *Sur Activation* dans le second.

Il y a un problème dans mon second cas : on ne peut avoir qu'une fenêtre active à la fois. Quand j'ouvre le formulaire *Livre*, ce dernier devient donc actif. Fermer *Livre* rend *Livres d'un auteur* de nouveau actif. Le moment serait donc bien choisi pour réactiver

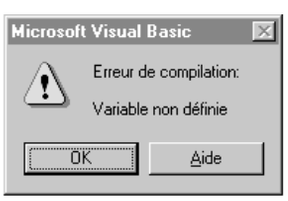
les boutons. Le problème, c'est que cliquer dans le formulaire *Livres d'un auteur* le rend actif alors même que *Livre* est affiché. Mes boutons redeviendraient disponibles, ce que je ne veux pas.
Seule la première solution est donc envisageable.
Ouvrez le formulaire *Livre* en mode *Création*, affichez ses propriétés et cherchez l'événement *Sur fermeture*. Affectez-lui *[Procédure événementielle]* puis cliquez sur le bouton … à côté.
Le squelette de la procédure événementielle est écrit. Il ne vous reste qu'à écrire le code :

```
Private Sub Form_Close()
  PremierEnreg.Enabled = True
  EnregSuivant.Enabled = True
  EnregPréced.Enabled = True
  DernierEnreg.Enabled = True
End Sub
```

Sauvegardez puis testez.

À la fermeture du formulaire *Livre*, on obtient… une erreur :

```
Private Sub Form_Close()
  PremierEnreg.Enabled = True
  EnregSuivant.Enabled = True
  EnregPréced.Enabled = True
  DernierEnreg.Enabled = True
End Sub
```



Que se passe-t-il encore ? Rien de grave ! C'était un piège ! Le message *Variable non définie* vous indique que *PremierEnreg* est inconnu. Eh oui ! Vous êtes actuellement dans le formulaire *Livre* qui ne possède aucun contrôle appelé *PremierEnreg*.
Il faut donc préciser qu'il s'agit du bouton *PremierEnreg* du formulaire *Livres d'un auteur*. Cela, vous savez le faire depuis le paragraphe 3A ! Corrigez votre code sans regarder la solution ci-dessous. Voici la solution :

```
Private Sub Form_Close()
  Forms![Livres d'un auteur].PremierEnreg.Enabled = True
  Forms![Livres d'un auteur].EnregSuivant.Enabled = True
  Forms![Livres d'un auteur].EnregPréced.Enabled = True
  Forms![Livres d'un auteur].DernierEnreg.Enabled = True
End Sub
```

Notez qu'il faut utiliser le nom du formulaire et non sa légende (soit le texte dans sa barre de titre en mode *Création* et non *Formulaire*).

Autre solution un peu optimisée, nous passons par une variable représentant le formulaire :

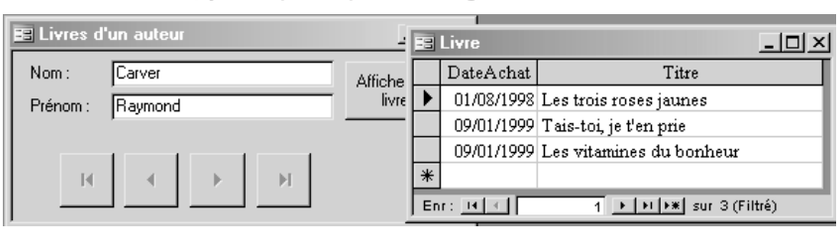
```
Private Sub Form_Close()
Dim f As Form
  Set f = Forms![Livres d'un auteur]
  f.PremierEnreg.Enabled = True
  f.EnregSuivant.Enabled = True
  f.EnregPréced.Enabled = True
  f.DernierEnreg.Enabled = True
End Sub
```

On peut faire encore mieux en utilisant la structure *With* de VB (utilisez l'aide si vous ne la connaissez pas). Cela donne le code qui suit.

```
Private Sub Form_Close()
  With Forms![Livres d'un auteur]
    .PremierEnreg.Enabled = True
    .EnregSuivant.Enabled = True
    .EnregPréced.Enabled = True
    .DernierEnreg.Enabled = True
  End With
End Sub
```

Testez : maintenant, cela fonctionne parfaitement !
Je choisis mon auteur :

Lorsque j'affiche ses livres, je ne peux plus changer d'auteur :



Lorsque je ferme la liste des livres, je peux à nouveau naviguer dans les auteurs :

Exercice

Chez moi, cela fonctionne. Et chez vous ? (Dit autrement, expérimentez.)

5. 2e exemple : vérification des données saisies

5A. Présentation

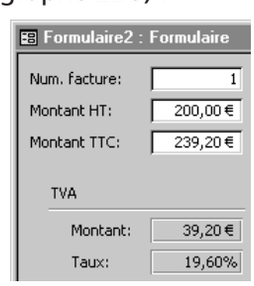
Nous allons maintenant nous pencher sur tout autre chose : comment faire pour vérifier des données saisies par l'utilisateur dans un formulaire ? Dans les premières séquences, nous avons abondamment étudié les propriétés *Valide si* et *Masque de saisie* des champs. C'est une première approche limitant les erreurs de saisie ; c'est insuffisant lorsque la vérification à effectuer est complexe. Il y a deux solutions :

- l'application calcule et affiche une donnée redondante. L'utilisateur est sensé vérifier cette donnée et corriger les données saisies s'il y a un problème ;
- l'utilisateur entre toutes les données, y compris la redondante. Si les données sont incohérentes, la saisie est refusée.

Bien entendu, il faut que les données aient un lien : impossible de vérifier un titre de livre par rapport au mois de naissance de l'auteur !
Nous allons étudier le cas déjà vu de la saisie des montants TTC et HT. L'idée est la suivante : je veux saisir une facture papier d'un fournisseur dans mon système informatique. Les montants TTC et HT sont liés avec le montant de TVA selon une relation classique que je vous rappelle : TTC = HT + TVA.

5B. Calcul et affichage de la donnée redondante

L'utilisateur saisit les montants TTC et HT. L'application affiche le montant de TVA (TVA = TTC – HT). L'utilisateur est alors sensé vérifier que ce montant est correct. Nous avions fait cela dans la séquence 9 (paragraphe 2D3) :



J'insiste : si le montant de TVA affiché ne correspond pas à celui indiqué sur la facture, l'utilisateur a fait une faute de frappe et doit corriger le montant HT ou TTC.
L'intérêt de cette technique ? On ne saisit que les informations nécessaires. L'inconvénient, c'est que l'on fait un peu trop confiance à l'utilisateur à mon goût : s'il ne vérifie pas les deux montants de TVA ou ne prend pas la peine de corriger les données (« bah, on n'est pas à 10 francs près »❶), les données saisies seront erronées.

❶ *Je vous jure que cette remarque est du vécu ! (D'où l'usage des francs et pas des euros.) Cela prouve la diversité du public des utilisateurs et l'obligation de paranoïa du développeur : ne faites* ***jamais*** *confiance à l'utilisateur.*

5C. Saisie de la donnée redondante et vérification de la cohérence

5C1. Montrons la solution

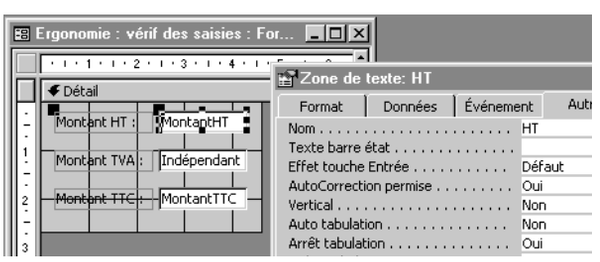
L'utilisateur saisit les montant TTC et HT mais aussi la TVA. L'application vérifie alors la cohérence des trois données et refuse la saisie si l'équation TTC = HT + TVA n'est pas vérifiée. Illustrons cela : juste après avoir saisi un montant TTC incohérent avec les autres montants, j'obtiens :

Attention, ce n'est pas forcément le montant TTC qui est faux : ce peut aussi être le montant HT ou TVA. Le seul message à afficher est donc que la relation entre les trois données n'est pas vérifiée.
L'inconvénient de cette solution est l'obligation pour l'utilisateur de rentrer beaucoup de données, dont une redondante. L'avantage, c'est qu'une saisie erronée n'est plus possible❶.

5C2. Comment ai-je fait cela ?

Comment ai-je fait cela ? Assez simplement : c'est de la programmation événementielle de base. Nous allons faire la manipulation.
Une fois le montant TTC saisi, je dois vérifier la cohérence des trois valeurs. Le « une fois le montant TTC saisi » doit vous faire penser à l'événement *Après MAJ* (pour *Mise À Jour*) de la zone de texte. Je vais donc écrire un peu de code lié à cet événement.
Avant tout, il faut nommer correctement les zones de texte. Comme je vais les manipuler par programmation, je ne peux me satisfaire des noms par défaut. J'appelle HT, TVA et TTC les trois zones de texte concernées (en modifiant la propriété *Nom* dans l'onglet *Autres*) comme le montre la copie d'écran suivante.

J'ai choisi de stocker les montants HT et TTC dans ma base. Les deux zones de texte correspondantes sont donc liées à la source de données.
En revanche, le montant de TVA n'est pas enregistré dans ma base puisqu'il ne sert qu'à la vérification. Il est donc affiché dans une zone de texte indépendante.



Je vais maintenant créer le code VBA de l'événement *Après MAJ* de la zone de texte *TTC*. L'algorithme est trivial :

si la relation entre les montants TTC, HT et TVA n'est pas vérifiée
  alors
    afficher un message d'erreur
fin-si

❶ *Et là, les avantages l'emportent largement sur les inconvénients, d'autant que l'utilisateur est déchargé de la phase fastidieuse de vérification des montants de TVA. Il gagne donc du temps.*

La seule difficulté réside dans l'écriture du test : comment exprimer que le montant TTC doit être égal au montant HT plus le montant de TVA ?
On ne peut pas écrire TTC = HT + TVA. En effet, *TTC*, *HT* et *TVA* sont des objets *zone de texte*. Cela n'a pas de sens de les ajouter ! Ce que l'on veut comparer, c'est le contenu de ces zones et non les zones elles-mêmes. Il me faut donc une propriété de l'objet *zone de texte* donnant la valeur de la zone. Cette propriété existe-elle ? Forcément ! C'est *Value*. (À distinguer de la propriété *Text*, consultez l'aide.)
De même, si je dis qu'une voiture est plus puissante qu'une autre, c'est un abus de langage : cette phrase signifie en fait que la puissance d'une voiture est plus élevée que la puissance d'une autre. Si j'ai deux objets véhicule *Voiture* et *Moto*, je ne peux pas dire :

Voiture < Moto

(D'ailleurs, que signifie cette comparaison ? Je compare le prix, la puissance, l'autonomie, l'agrément de conduite, l'âge des véhicules ?)

En revanche, je peux comparer les propriétés des objets :

Voiture.Poids < Moto.Poids

Ainsi, on va dire que le montant TTC saisi (*TTC.Value*) doit est égal au montant HT (*HT.Value*) plus le montant de TVA (*TVA.Value*).

Au final, on aura le code suivant :

```
Private Sub TTC_AfterUpdate()
 Dim Bidon As Integer
 If HT.Value + TVA.Value <> TTC.Value Then
   Bidon = MsgBox("Le montant TTC doit être égal au montant HT augmenté de la TVA.", _
                  vbOKOnly + vbCritical, "Erreur de saisie")
 End If
End Sub
```

Quelques explications :

- notez bien le nom de la fonction : c'est le nom du contrôle (*TTC*) suivi d'un tiret et du nom de l'événement (*Après MAJ* se dit *AfterUpdate* en anglais) ;
- pour une explication éventuelle sur la fonction *MsgBox*, allez voir l'aide ;
- comme *MsgBox* est une fonction, il me faut récupérer son résultat dans une variable : c'est une obligation grammaticale ; cela dit, comme cette valeur ne me sert à rien, j'utilise une variable *Bidon* pour bien marquer mon mépris.

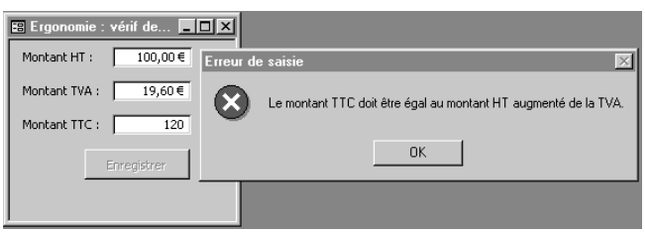
5C3. Finalisons !

C'est presque fini. Un petit problème reste en suspens : je vous ai dit ci-dessus que l'application contrôlait les trois valeurs saisies et refusait l'ensemble si la relation TTC = HT+ TVA n'était pas vérifiée.
Mais comment faire pour interdire la saisie ? L'idée est de mettre un bouton dans le formulaire. Il sera chargé d'enregistrer la saisie dans la base. En fait, notre code de vérification activera le bouton si le test est vérifié et le désactivera sinon. Le bouton étant appelé *Enregistrer*, j'aurai le code page suivante :

```
Private Sub TTC_AfterUpdate()
  Dim Bidon As Integer
  If HT.Value + TVA.Value <> TTC.Value Then
      Enregistrer.Enabled = False
      Bidon = MsgBox("Le montant TTC doit être égal au montant HT augmenté de la TVA.", _
                     vbOKOnly + vbCritical, "Erreur de saisie")
    else
      Enregistrer.Enabled = True
  End If
End Sub
```

Voici deux exemples.

Les montants n'étant pas cohérents, le bouton d'enregistrement n'est pas accessible.



Les montants saisis sont cohérents... on peut enregistrer.

5C4. Optimisons !

Réfléchissez maintenant à ce que nous avons fait. Est-ce parfait ? Supposons que les nombres saisis soient incohérents : le message d'erreur s'affiche. Vous devez donc corriger les valeurs. Si c'est le montant HT qui est faux, vous le corrigez. Que devrait-il alors se passer ?

Exemple

Vous avez rentré les valeurs suivantes :

Première saisie	
HT	10,00 €
TVA	19,60 €
TTC	119,60 €

Comme 119,6 n'est pas égal à 10+19,6, vous obtenez le message d'erreur. En fait, il ne fallait pas rentrer 10 € HT mais 100 € HT ! Vous corrigez donc :

Deuxième saisie		
HT	**1000,00 €**	← **Oups !**
TVA	19,60 €	
TTC	119,60 €	

Hé oui ! Tête de linotte que je suis… j'ai encore fait une faute de frappe ! Et là, que va-t-il se passer ? Rien, puisque ma procédure de vérification ne se déclenche que lorsque je modifie la valeur du montant TTC.
Or, le message d'erreur indique bien que l'une des valeurs est erronée et pas nécessairement le montant TTC. Bref, chaque saisie peut potentiellement provoquer une erreur. La première saisie est assez naturellement faite de haut en bas, soit HT, TVA puis TTC. Il est donc normal de lancer la vérification sur la saisie du montant TTC. En revanche, si elle est erronée, l'utilisateur va modifier un des trois montants. Il faut alors de nouveau vérifier la saisie (de toute façon, on ne peut pas se baser sur un ordre de remplissage *naturel* : l'utilisateur peut préférer rentrer les données de bas en haut !).

Comment faire ? Il y a deux solutions :

- soit chaque modification de l'un des trois montants déclenche la vérification. Il faut alors associer notre procédure de vérification aux trois événements *Sur_MAJ* des zones de texte ;
- soit la vérification est faite lorsque l'utilisateur demande spécifiquement à enregistrer… bref, c'est le bouton *Enregistrer* qui vérifie les valeurs avant d'enregistrer si tout va bien.

Ici, les deux choix sont équivalents car le formulaire est tout petit. D'une façon générale, on essaye de vérifier les données au plus tôt, à savoir dès leur saisie car plus tôt l'utilisateur sait qu'il a fait une erreur, mieux c'est. Supposez qu'après avoir saisi mes trois montants, je saisisse les coordonnées du débiteur et le contenu de la facture. Cela fait beaucoup d'informations ! Je clique enfin sur *Enregistrer* et j'apprends que l'ensemble de ma saisie est invalide car les trois montants sont incohérents. C'est assez frustrant ! (Et encore, dans ce cas simple, c'est dû à une faute de frappe de ma part donc je peux corriger la saisie.)
Nous allons donc utiliser la première solution : la modification de l'un des trois montants entraîne la vérification de l'ensemble. C'est toujours la même procédure qui est exécutée. La pire solution serait de l'écrire trois fois :

```
Private Sub HT_AfterUpdate()
  Dim Bidon As Integer
  If HT.Value + TVA.Value <> TTC.Value Then
    Enregistrer.Enabled = False
    Bidon = MsgBox("Le montant TTC doit être égal au montant HT augmenté de la TVA.", _
                   vbOKOnly + vbCritical, "Erreur de saisie")
    Else
      Enregistrer.Enabled = True
  End If
End Sub
Private Sub TTC_AfterUpdate()
  Dim Bidon As Integer
  If HT.Value + TVA.Value <> TTC.Value Then
    Enregistrer.Enabled = False
      Bidon = MsgBox("Le montant TTC doit être égal au montant HT augmenté de la TVA.",_
                     vbOKOnly + vbCritical, "Erreur de saisie")
    Else
```

```
        Enregistrer.Enabled = True
    End If
End Sub
Private Sub TVA_AfterUpdate()
    Dim Bidon As Integer
    If HT.Value + TVA.Value <> TTC.Value Then
        Enregistrer.Enabled = False
        Bidon = MsgBox("Le montant TTC doit être égal au montant HT augmenté de la TVA.", _
                        vbOKOnly + vbCritical, "Erreur de saisie")
    Else
        Enregistrer.Enabled = True
    End If
End Sub
```

Là, c'est un problème d'algorithmique (et de bon sens) : lorsque l'on utilise plusieurs fois le même code, on ne l'écrit pas plusieurs fois : on le place dans un sous-programme que l'on appellera autant de fois que nécessaire.
Nous allons donc créer une procédure *VérificationMontants* qui sera appelée par les trois procédures événementielles :

```
Private Sub VérificationMontants()
    Dim Bidon As Integer
    If HT.Value + TVA.Value <> TTC.Value Then
        Enregistrer.Enabled = False
        Bidon = MsgBox("Le montant TTC doit être égal au montant HT augmenté de la TVA.", _
                        vbOKOnly + vbCritical, "Erreur de saisie")
    Else
        Enregistrer.Enabled = True
    End If
End Sub
```

```
Private Sub HT_AfterUpdate()
    VérificationMontants
End Sub
```

```
Private Sub TTC_AfterUpdate()
    VérificationMontants
End Sub
```

```
Private Sub TVA_AfterUpdate()
    VérificationMontants
End Sub
```

5C5. Dernier réglage (promis !)

Avons-nous enfin fini ? Presque ! Imaginons que le formulaire vienne d'être affiché ; je commence la saisie en rentrant le montant HT. Les autres montants ne sont pas encore entrés. Cela n'a donc pas de sens de vérifier dès maintenant la cohérence des trois nombres puisqu'un seul est saisi.
Comment régler cela ? C'est simple : on ne fera la vérification de la cohérence que lorsque l'on aura trois valeurs à vérifier, soit lorsque les trois zones de texte seront remplies.
Maintenant, comment savoir si une zone de texte est vide ou remplie ? On a accès au mot réservé *Empty* : si la zone *Z* est vide, alors *Z.Value* = *Empty* (ce qui signifie intuitivement que le contenu de *Z* est *vide*).
On ne veut faire le test que lorsque les trois zones sont remplies, donc lorsqu'aucune n'est vide. Ainsi, si la zone *HT* **n'est pas** vide **et** la zone *TVA* **n'est pas** vide **et** la zone *TTC* **n'est pas** vide, alors on vérifie leur cohérence.

Cela donnera :

```
Private Sub VérificationMontants()
Dim Bidon As Integer
If HT.Value <> Empty And TVA.Value <> Empty And TTC.Value <> Empty Then
    If HT.Value + TVA.Value <> TTC.Value Then
      Enregistrer.Enabled = False
      Bidon = MsgBox("Le montant TTC doit être égal au montant HT augmenté de la TVA.",_
                     vbOKOnly + vbCritical, "Erreur de saisie")
      Else
        Enregistrer.Enabled = True
    End If
  End If
End Sub
```

Voilà ! Cette fois, c'est terminé. Vous remarquerez qu'un problème somme toute simple nous entraîne assez loin ! Et ce n'est qu'en testant notre réalisation que l'on s'est rendu compte des différentes erreurs.

Nous n'irons pas plus loin pour le moment (nous reprendrons brièvement ce dernier formulaire dans la séquence 13 sur l'ergonomie pour apprendre à laisser une porte de sortie).
Relisez cette partie en diagonale : une fois le principe assimilé, nous n'avons fait que de l'algorithmique de base : des tests et des affectations. Un peu de code VB associé à la manipulation de propriétés simples change vraiment la vie d'une application.
C'est rentable : rapide à faire et très intéressant au niveau de l'ergonomie.

Exercice

Dois-je vraiment vous le dire ?
Une fois que vous avez lu et compris ces manipulations, revenez au début du 2e exemple et faites les manipulations.

Synthèse

Finalement, je ne devais faire que quelques pages, et bon… je préfère ne pas les compter. Cela dit, nous avons vu des choses très intéressantes :

- le côté VB du code VBA ;
- l'apport de VBA (ses objets métiers) ;
- la notion d'objet (propriétés et méthodes) ; n'oubliez pas que l'affectation d'un objet se fait avec l'opérateur *set* ;
- comment générer du code avec l'assistant et rajouter ou modifier soi-même quelques instructions pour optimiser le fonctionnement de l'application.

Bien entendu, il vous restera l'année prochaine à apprendre les traitements *lourds* : manipuler tables, requêtes et enregistrements par programmation.

Cela dit, ce cours Access vous permet dès à présent de réaliser des bases de données d'un bon niveau… surtout si vous mettez à profit la dernière séquence !

Séquence 13

Ergonomie

Vous avez en main toutes les connaissances techniques pour réaliser une application Access. Cette séquence va vous fournir les règles de base permettant de réaliser des applications agréables à utiliser et intuitives.

▶ *Capacités attendues*

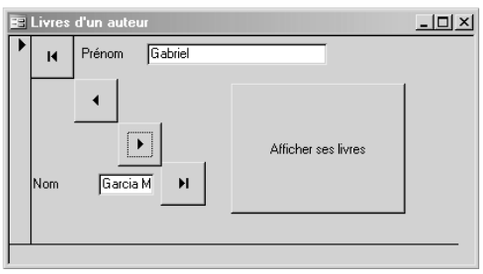
- **Réaliser des applications d'aspect professionnel avec un langage de développement visuel**

▶ *Contenu*

1. Introduction

1A. Exemple

Comparez les trois formulaires suivants :



Fonctionnellement, ils sont **identiques**. Hé oui ! En effet, tous trois :

- affichent le nom et le prénom de l'auteur courant ;
- permettent de naviguer dans les auteurs ;
- affichent sur demande les livres de l'auteur courant.

Je ne vous cache pas que pour le troisième, j'ai dû rajouter du code afin que ces traitements soient réalisés ! Pour autant, ces trois formulaires sont-ils aussi pratiques d'emploi ?

- le premier, vous le connaissez ; c'est celui que nous avons réalisé dans la séquence précédente. Il n'est pas particulièrement beau, mais il est lisible (sobre et efficace). En quelques instants, vous vous l'appropriez ;
- le second est une version créative du premier. Les boutons de déplacement sont en escalier, celui affichant les livres est monstrueux et le nom et le prénom sont aux

deux extrêmes. De plus, la zone contenant le nom est trop petite. Ce formulaire n'est pas évident à assimiler !

- pour le troisième formulaire, j'ai débridé tous mes instincts : j'ai placé Nina le petit Teckel sympa et Niok le chat. Ensuite, les zones de texte contenant le nom et le prénom de l'auteur ne sont pas alignées et j'ai utilisé des contrôles et des événements inadaptés (liste déroulante pour réaliser un traitement, survol du bouton). Enfin, j'ai mis des blagues nulles. Même si c'était de l'humour de qualité (et j'ai bien pris garde à ce que cela ne soit pas le cas), que penser de l'effet comique lorsque l'utilisateur voit le formulaire pour la 500e fois ?

Lorsque vous utilisez Word ou Excel, vous ne devez pas avoir trop de difficultés car :

- l'application elle-même n'est pas très technique ;
- les différentes fenêtres que l'application affiche sont intuitives et compréhensibles.

De même, bien qu'Access soit un outil complexe nécessitant des connaissances théoriques, les assistants sont très bien faits et dans une certaine mesure vous permettent de réaliser des choses que vous ne sauriez pas faire. Et cela, c'est l'ergonomie.

1B. Pourquoi l'ergonomie

L'utilisateur de l'outil informatique (donc des logiciels) n'est pas un interlocuteur agréable pour le développeur. En effet, ce dernier est fier de ses algorithmes, des optimisations qu'il a su réaliser… L'utilisateur, lui, ne sait même pas ce qu'est un algorithme. Pour lui, un programme se résume à ce qui est affiché à l'écran ou sort sur l'imprimante.
Je veux dire par là qu'un programme techniquement parfait mais pas pratique à utiliser sera beaucoup moins bien perçu qu'un programme moins efficace mais d'un abord facile et agréable.
Et c'est là où le bât blesse : vous êtes l'utilisateur le plus averti de votre application donc vous n'avez pas besoin d'aide ou de documentation ! En effet, comme c'est vous qui l'avez faite, vous la maîtrisez dans ses moindre recoins. Vous avez structuré l'application comme vous l'entendez, elle correspond donc à votre schéma mental.
En revanche, l'utilisateur connaît plus ou moins les fonctionnalités de l'application puisqu'il vous les a commandées mais sa logique ne lui est pas forcément évidente : si pour vous il est naturel de passer par les clients pour avoir les factures, l'utilisateur peut préférer passer par les commandes. Parfois, c'est l'utilisateur qui a raison (c'est donc vous qui avez mal conduit votre analyse), parfois c'est vous : un des rôles de l'informatique est de remettre à plat et corriger les règles de gestion de l'entreprise.
Dans la réalité, bien entendu, il faut éviter de livrer un produit terminé à l'utilisateur tel le magicien sortant un lapin de son chapeau. Vous l'associerez au développement en décidant avec lui de l'aspect des formulaires et de l'ordre dans lequel ils doivent s'enchaîner.
Cela dit, l'utilisateur n'ayant pas d'expérience dans la conception, c'est à vous de guider ses choix et de proposer des choses correctes lui permettant d'assimiler le fonctionnement de l'application de la façon la plus simple possible.

1C. Définition

Pour cela, des règles précises existent : on parle de règles d'ergonomie. Mais au fait, l'ergonomie, c'est quoi ? *Le Petit Robert* nous répond : l'ergonomie, c'est « l'étude scientifique des conditions (psychophysiologiques et socioéconomiques) de travail et des relations entre l'homme et la machine ».

En pratique, les règles d'ergonomie vont nous permettre d'optimiser la relation entre l'application et l'utilisateur. Et cette relation, c'est ce que l'on appelle l'interface (*Le Petit Robert* : l'interface [nom féminin] est la « Jonction permettant un transfert d'informations entre deux éléments d'un système informatique »). Ici, les deux éléments sont l'application d'un côté et l'utilisateur de l'autre. L'interface est donc avant tout l'affichage à l'écran (soit les formulaires). Cela se divise en deux :

- comment les informations sont affichées. C'est le choix des contrôles ;
- comment l'utilisateur peut agir (lancer un traitement, saisir des informations…). C'est le choix des événements.

Finalement, l'ergonomie est partout : dans le traitement de texte (faites des documents sobres et élégants), le tableur (soignez vos feuilles de calcul avec l'emploi des formats d'affichage). La seule différence, c'est que l'ergonomie prend une dimension plus grande en développement.

2. Règles d'ergonomie : principes de base

J'insiste : autant vous pouvez faire ce que vous voulez avec le code (sous-entendu, le rendre le plus efficace possible), autant l'interface n'est pas votre chasse gardée mais plutôt celle de l'utilisateur.
Cela dit, comme l'utilisateur n'a pas de recul, c'est vous qui devrez l'interroger pour comprendre ce qu'il attend (comment les données doivent être organisées, comment faire la liaison entre les formulaires…). Sur ces bases, vous ferez des maquettes de formulaire puis vous les soumettrez à l'utilisateur jusqu'à obtenir un consensus.
Lorsque vous soumettrez des exemples de formulaires, pas question de proposer le 2e ou le 3e formulaire vu dans le paragraphe 1A. Comprenez bien que si vous ne montriez que le 3e formulaire, l'utilisateur n'aurait pas les connaissances pour penser qu'on peut mieux faire ! Il se dirait juste que l'informatique, décidément, c'est tordu.

Plus votre application sera simple d'emploi, plus l'utilisateur sera motivé pour l'utiliser. Et faire une application simple, c'est difficile ! Cela vous oblige à beaucoup de travail pour simplifier et optimiser vos formulaires.
Comment faire pour que l'utilisateur apprécie votre application, c'est-à-dire la trouve simple d'emploi ? Voici quelques règles en vrac :

- il faut éliminer ou au moins réduire le plus possible les bugs (les erreurs de développement) ;
- vous devez limiter au maximum l'impact des fausses manœuvres éventuelles. Dans la séquence précédente, nous avons désactivé les boutons de déplacement lorsque l'utilisateur ne devait pas s'en servir. En pratique, vous désactiverez toutes les commandes qui ne doivent pas être utilisées à un moment donné. L'utilisateur n'aura donc pas tendance à les lancer pour *voir ce que cela fait*. Je vous assure qu'activer puis désactiver en fonction de l'exécution, c'est fastidieux. Il faut pourtant le faire ;
- l'interface doit être la plus simple et lisible possible : n'affichez que les informations utiles, utilisez le bon contrôle, proscrivez les fautes de français…
- surtout, vous devez vous fondre dans le moule : il faut que l'utilisateur trouve ses marques dans votre application comme il la trouve dans son traitement de texte, son tableur… Il faut adopter les mêmes conventions (*F1* pour l'aide), les mêmes menus (menu *Fichier/Enregistrer*…). Il ne s'agit pas de faire le mouton de Panurge, mais juste de ne pas briser l'habitude des gens. Il est plus simple de retrouver la même interface dans les différents logiciels. Par exemple, tout le monde connaît le

bouton de la barre d'outils contenant le dessin d'une disquette. Il sert à enregistrer le fichier courant. L'utilisateur voyant ce bouton dans votre application saura à quoi il sert sans avoir à consulter l'aide. Un sérieux gain de temps !

Avant Windows, les écrans étaient en mode caractère (sous Dos). Un des intérêts de Windows a été l'uniformisation des interfaces. J'ai appris l'informatique en mode Dos, où l'anarchie régnait : j'avais deux programmes qui utilisaient la touche *F11* différemment : l'un lançait l'aide, l'autre se terminait… combien de fois j'ai quitté ce dernier en voulant lancer l'aide !
En fait, les deux cas très précis ou vous pouvez (voire devez) être original et ne pas suivre les règles habituelles sont lorsque vous programmez :

- un jeu. L'interface définissant l'atmosphère, il serait idiot de conserver celle de Windows ;
- un système d'exploitation. En effet, c'est lui qui détermine l'ergonomie des applications qu'il fait tourner.

Nous allons maintenant étudier les règles d'ergonomie relatives aux différents concepts du formulaire. Je ne peux faire autrement que vous donner une liste de règles à respecter. À chaque fois que possible, je vous fournirai des exemples concrets. N'hésitez pas à consulter les fenêtres des diverses applications (Word, Excel, VB, Delphi…) pour avoir des idées de bonnes interfaces.

3. Ergonomie du formulaire : aspect visuel

3A. Organisation des formulaires

3A1. Présenter les informations dans un formulaire

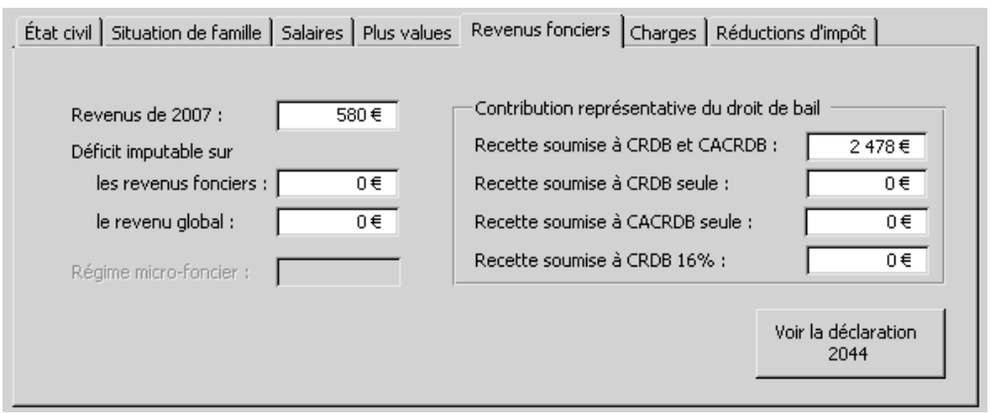
Un formulaire permet d'afficher et/ou de saisir des informations. Observez votre feuille de déclaration de revenus. C'est un formulaire papier réputé pour sa densité. Son ergonomie n'est pas terrible mais dans ce cas très précis, d'autres règles ont été jugées plus importantes : faire un document lisible et aéré aurait donné une liasse de cinquante pages, ce qui aurait posé d'autres problèmes (manipulation, encombrement, transport…).

Si nous devions faire un formulaire pour saisir la déclaration de revenus, une technique consisterait à recopier la présentation de la déclaration papier.

En fait, il vaudrait mieux plusieurs formulaires, un par rubrique (par exemple, un pour les renseignements d'état civil, un pour les salaires, un pour les déductions…). Bien entendu, l'enchaînement des différents formulaires (la navigation de l'un à l'autre) sera à votre charge ; idem pour la récupération des données.

Une autre solution, préférable dans ce cas, consisterait à utiliser un onglet par rubrique dans la déclaration.

Voici un exemple :



3A2. Répartir les informations sur divers formulaires

La déclaration est faite sur le formulaire papier n° 2042. Ce formulaire contient diverses rubriques : état civil, situation de famille, salaires, revenus fonciers… Dans certains cas, il faut déposer – en plus de la déclaration 2042 – d'autres déclarations complémentaires détaillant certaines rubriques.
Par exemple, les revenus fonciers se calculent en ajoutant tous les revenus (loyers, provisions sur charges, subventions) et en soustrayant toutes les charges (emprunt, dépenses locatives, taxe foncière, travaux…). La différence entre les revenus et les charges donnera lieu, après encore quelques calculs, au revenu foncier imposable. Le formulaire 2044 permet d'expliciter tous les calculs aboutissant au revenu foncier. C'est ce chiffre que l'on reportera dans la déclaration 2042. Ici, je déclare donc pour 580 € de revenus fonciers. Les 2 478 € correspondent aux loyers bruts encaissés (j'ai établi ce chiffre grâce à ma précieuse déclaration 2044).

Pour informatiser cela, deux solutions : soit je mets toutes les informations et les calculs dans un seul formulaire (évidemment illisible), soit je structure. Ici, je structure :

- l'ensemble de ma déclaration 2042 est reprise dans un formulaire (présenté ci-dessus), chaque rubrique étant placée dans un onglet ;
- pour une rubrique (donc un onglet) donnée, un bouton permet d'afficher la déclaration complémentaire correspondante. Ici, la rubrique *Revenus fonciers* possède un bouton permettant d'afficher la déclaration 2044.

Ce qu'il faut retenir, c'est que j'ai finalement trois niveaux : le formulaire principal, les onglets et, dans chaque onglet, un bouton si besoin est pour afficher une autre déclaration. Bien entendu, cette autre déclaration sera elle aussi structurée en onglet si nécessaire. L'objectif est de n'avoir qu'une dizaine de contrôles affichés à la fois.
Une autre technique permettant de structurer les informations, c'est le sous-formulaire étudié dans la séquence 6 : au lieu d'afficher les informations de tous les livres et tous les auteurs, on va afficher les auteurs un par un, un sous-formulaire donnant tous les livres de l'auteur courant.

3B. Couleur

Lorsque je lance mes étudiants sous Access, ils l'étudient dans le même ordre peu original que vous (tables, requêtes, formulaires, états…). Tout va bien jusqu'à la requête. Mais alors, il suffit que je tourne la tête quelques secondes lorsqu'ils abordent les formulaires pour me retrouver dans mon jardin : les formulaires fleurissent en rose, en jaune lumi-

neux, en violet, en vert qui flashe… Bref, c'est comme si mes étudiants avaient été brimés par la grisaille des écrans Windows et découvraient la couleur.

Pourquoi les développeurs de Microsoft ont-ils été moins inspirés que mes étudiants et se sont limités au fond gris et encre noire ? Ai-je dans ma classe des développeurs géniaux qui révolutionneront les interfaces graphiques ? Ma foi non…
Le choix des couleurs est quelque chose de tout à fait technique qui ne laisse aucune place au hasard ou au bariolage :

- des couleurs vives ou non assorties fatiguent l'œil. Je ne pense pas que vous teniez trois heures face à un écran *panthère rose*. Il faut donc utiliser des couleurs douces et les associer les unes aux autres pour avoir un ensemble cohérent et homogène. Bien entendu, plus vous employez de couleurs, plus il est difficile de conserver une certaine unité ;
- il vaut mieux utiliser des teintes chaudes et engageantes que des teintes froides ;
- les couleurs sont perçues de façon très subjective : ce qui semble joli à quelqu'un sera inesthétique aux yeux d'un autre ; il n'aimera donc pas utiliser votre application. C'est pire encore lorsque votre application est internationale car les codes de couleur varient d'un pays à l'autre ;
- les couleurs permettent de mettre des éléments en valeur. Tout mettre en valeur n'a pas grand sens.

Au final, on retrouve mon avertissement sur les mises en forme sous Word : le débutant aura naturellement tendance à en faire trop. Les développeurs de Microsoft ont utilisé des fonds gris, ce qui semble au premier abord le pire choix ! On peut donc se douter que s'ils ont malgré tout retenu ce coloris, c'est après une étude sérieuse.

3C. Les menus

Si vous faites des menus (barre de menus) ou des barres d'outils, vous respecterez les points suivants :

- distinguez (barres d'outils différentes, menus différents) les commandes classiques (sauvegarde, impression, aide…) des commandes propres à votre application ;
- vous regrouperez toutes les commandes de manipulation de votre document (sauvegarde, impression…) dans un menu *Fichier* ;
- toutes les commandes de texte (copier/coller…) seront dans le menu *Édition* ;
- le dernier menu sera « ? » et correspond à l'aide.

Bref, calquez-vous autant que possible sur les menus de Word ou d'Excel. Si vous autorisez les raccourcis clavier, conservez ceux normalisés (*F1* pour l'aide, *Ctrl+C* pour *Copier*…).
Les commandes très particulières qui n'ont un sens que dans une partie de l'application seront placées dans des menus contextuels accessibles par clic-droit. Pour plus d'informations sur ces menus, allez voir l'aide d'Access ou de VB.

4. Ergonomie des contrôles

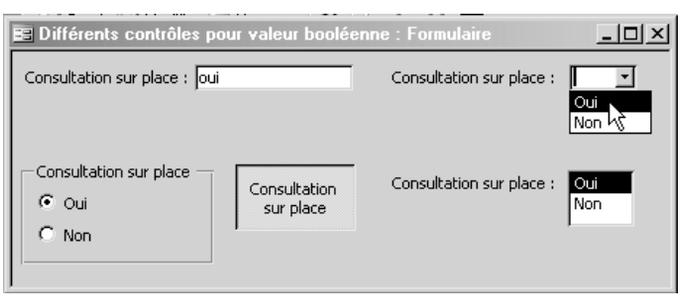
Il s'agit toujours des contrôles du formulaire. J'en ai fait un paragraphe distinct car il y a beaucoup à dire.

4A. *Choix du contrôle*

Le choix des contrôles représente la majeure partie de l'ergonomie. Nous les avons tous étudiés dans la séquence 8. Je ne les détaillerai pas de nouveau. Comprenez bien que chaque contrôle a son rôle. Ils ne diffèrent pas uniquement par leur aspect visuel.
Prenez ce cours en main et allez dans votre cuisine… c'est bon ? Vous y êtes ? Non, vous êtes toujours assis à votre bureau. Allez dans votre cuisine, je vous dis !
Bien. Regardez maintenant autour de vous. Vous voyez différents outils : des poêles, des casseroles, des couverts, des ustensiles (fouet, presse-ail, mixer…). Chaque outil ne ressemble évidemment pas aux autres. Cependant, ce n'est pas son aspect physique qui compte mais sa fonction❶. Lorsque vous voulez faire quelque chose, vous prenez l'outil adéquat, celui dont la fonction correspond à ce que vous voulez faire :

- techniquement, vous pouvez faire cuire du riz dans une louche : vous mettez de l'eau dans la louche, vous tenez cette dernière au-dessus d'une flamme et quand l'eau bout, vous y placerez deux ou trois grains de riz. Au bout d'une semaine, vous aurez assez de riz pour faire un bon repas ;
- autre approche, vous employez l'outil adéquat : une casserole.

Vous pouvez retourner dans votre bureau.
Que faut-il retenir de cette visite à la cuisine ? Et bien, quelqu'un de très débrouillard pourra faire tout et n'importe quoi avec deux bouts de ficelle et trois allumettes. Cela dit, il est plus agréable et simple d'avoir l'outil adapté à la tâche.
C'est la même chose avec les contrôles. Pour saisir une valeur booléenne, toutes les solutions suivantes sont techniquement acceptables :



J'ai utilisé une zone de texte, un bouton bascule, un groupe d'options, une liste déroulante et une liste. Ces contrôles sont *techniquement acceptables* car ils permettent plus ou moins efficacement de saisir une valeur booléenne (*oui* ou *non*). Ces solutions ne sont en revanche pas pratiques du tout :

- toutes obligent à réaliser des manipulations et prennent beaucoup de place dans le formulaire pour réaliser la saisie d'une donnée élémentaire ;
- la zone de texte est le pire choix car l'utilisateur peut rentrer n'importe quoi. Comment réagira votre application si l'utilisateur saisit « Oui, enfin… je crois. » ou « Yes » ? Évidemment, vous pouvez écrire du code VBA réagissant à l'événement *après MAJ* et vérifiant que la saisie est soit *oui*, soit *non* (problème de majuscules mis à part). Et là, cela devient vraiment lourd.

❶ *D'ailleurs, c'est la fonction qui détermine l'aspect physique et non l'inverse.*

La seule solution acceptable est la case à cocher :

C'est parfait car je n'ai qu'un clic à faire, je suis sûr de ne pas saisir de valeur erronée (ce sera soit *oui*, soit *non*) et ce contrôle occupe un minimum de place.
Vous pourriez être tenté d'employer un bouton bascule qui correspond à la même fonction. Je vous le déconseille : vous ne trouverez ce bouton dans aucune application sérieuse. Je vous assure qu'il est moins visuel que la case à cocher. Dans ce cas là, pourquoi Access le propose-t-il ? Ma foi, Word propose bien le souligné…

N'hésitez pas à retourner dans la séquence 8 relire attentivement ma description des contrôles : maintenant que vous maîtrisez le point de vue technique, vous pourrez vous concentrer sur leur emploi.

4B. Libellés (étiquettes) des contrôles

Ici encore, pas question de faire n'importe quoi : les libellés sont considérés comme des phrases, donc une majuscule au début de chacun d'eux. Ils sont tous terminés par un « : », sauf pour la case à cocher et le groupe d'options (voir la copie d'écran du paragraphe précédent).
Bien entendu, une faute d'orthographe est absolument catastrophique sur du texte que l'utilisateur aura constamment sous les yeux.

4C. Position des contrôles

4C1. Principe

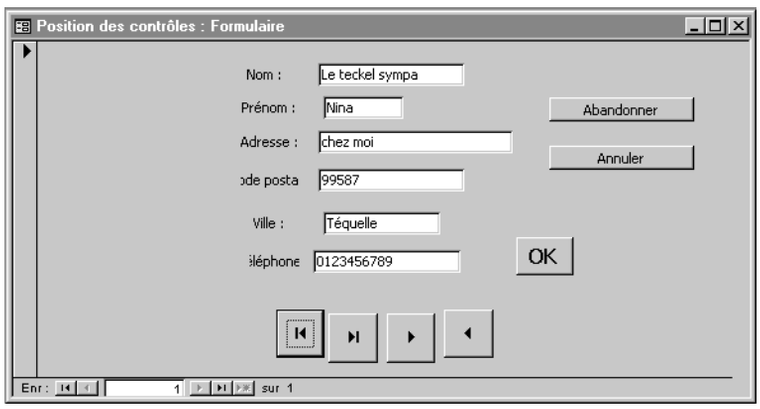
Comment positionner les contrôles dans le formulaire ? Ils ne se placent pas aléatoirement, l'ordre le plus strict doit régner. Ordre et discipline, soldat !
Retournez voir les trois formulaires du paragraphe 1A. Dans le premier, les contrôles sont bien positionnés. Dans les autres, non : ni les zones de texte ni les boutons ne sont bien placés.
Le principe est simple :

- les boutons doivent avoir la même taille et être alignés à gauche (s'ils sont positionnés verticalement) ou en haut (positionnés horizontalement) ;
- les zones de texte doivent être de même longueur (pour des saisies de même nature). Les libellés (étiquettes) des zones de texte seront alignées à gauche (bref, l'écart entre étiquette et zone de texte est variable selon la longueur du libellé) ;
- les écartements verticaux et horizontaux des contrôles doivent être constants.

Le pire, c'est lorsque les contrôles ne sont pas tout à fait alignés, qu'il s'en faille de quelques pixels. Car alors, l'utilisateur a une impression de gène diffuse, sans trop savoir d'où elle vient.

4C2. Exemple

Voici un exemple de formulaire qui ne va pas du tout. Avant de lire la suite (détaillant les erreurs et corrigeant le formulaire), essayez d'identifier tout ce qui ne va pas.



C'est bon ? Vous avez votre petite liste d'erreurs ?

Et bien, voici la mienne :

- les contrôles sont décalés sur la droite, ce qui fait que la partie gauche du formulaire est vide ;
- les zones de texte ont des tailles aléatoires et ne sont pas alignées à gauche ; de plus, leur écartement vertical est variable ;
- cela fait partie de l'ergonomie même si je ne l'ai pas abordé dans cette séquence, il faut utiliser un masque de saisie pour afficher correctement le téléphone ;
- les libellés des zones de texte sont centrés et pas alignés à gauche. Ceux du téléphone et du code postal sont tronqués car trop petits ;
- les boutons *Abandonner*, *Annuler* et *OK* ne sont pas alignés ni de même taille ni de même style (*OK* est en gras) ;
- il y a un problème avec *Abandonner* et *Annuler* : à priori, c'est la même chose. Si les deux boutons ont réellement une fonction différente, il faut changer leur libellé pour les rendre plus explicites.
- les boutons de déplacement ne sont pas alignés et l'écart entre eux n'est pas constant ; de plus, ils sont dans un ordre tout à fait inhabituel !
- si l'on a rajouté des boutons de déplacement, il est inutile de laisser la barre d'état du formulaire.

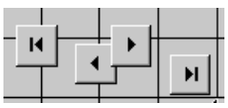
Vous remarquerez qu'il est très fastidieux de positionner correctement les contrôles. C'est pourquoi Access peut le faire pour vous : en mode *Création*, sélectionnez plusieurs contrôles puis dans le menu *Format*, essayez les commandes *Aligner…*, *Taille*, *Espacement horizontal* et *Espacement vertical*. Dans *Format*, vous remarquerez également *Grouper*. Cela permet de grouper plusieurs contrôles. On peut alors les déplacer en groupe sans que leurs positions respectives ne change. Très pratique une fois que les contrôles sont bien positionnés ! Bien entendu, on peut supprimer le regroupement avec *Format/Dissocier*.

4C3. Corrigeons le formulaire

Nous allons corriger ce formulaire. Avec le menu *Format*, c'est très rapide !

1. Les boutons de déplacement.

Je les replace dans le bon ordre :



Je ne me soucie pas de leur position puisque les outils du menu *Format* s'en chargeront !

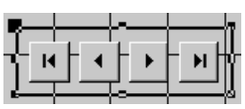
Je les sélectionne tous :

Comment faire ? Je fais *Shift+Clic* sur chacun d'eux ou alors je clique dans l'angle supérieur gauche, je laisse appuyé et descend jusqu'à l'angle inférieur droit (technique du lasso).

Menu *Format/Aligner/Haut* :

Menu *Format/Espacement horizontal/Égaliser* :

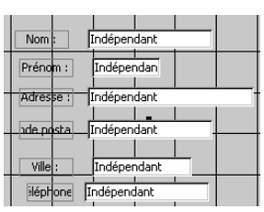
Menu *Format/Grouper :*



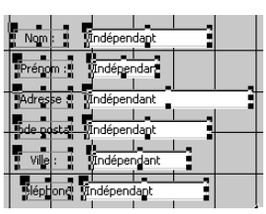
Reconnaissez que c'était vite fait !

2. Espacement des zones de texte.
Je les sélectionne complètement (zone et libellé) puis *Format/Espacement Vertical/Égaliser*.

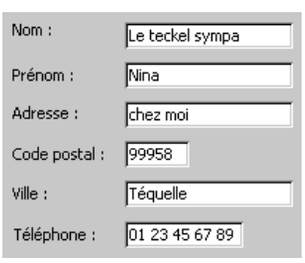
Avant



Après

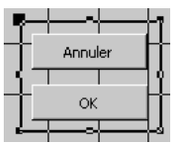


3. De même, je redimensionne les libellés et je les aligne à gauche (en prenant soin de ne sélectionner que les libellés et pas les zones de texte). Tant que j'y suis, j'uniformise la taille des zones de texte, je les aligne aussi à gauche (en ne sélectionnant qu'elles), j'applique un masque pour la saisie du téléphone et enfin, j'égalise l'espacement vertical. Une fois tout cela fait, je groupe les zones de texte pour pouvoir ultérieurement les déplacer en bloc.

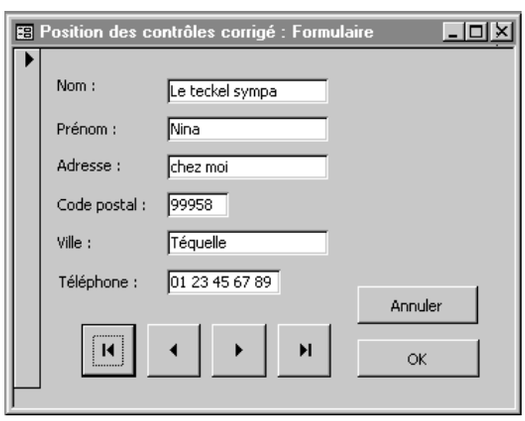


Vous remarquerez que pour le code postal et le téléphone, les zones sont plus petites car les données sont d'une taille limitée et faible. Il n'aurait pas été beau de les faire aussi longues que les autres.

4. Je supprime le bouton *Abandonner* qui était redondant avec *Annuler*. Je redimensionne et déplace *OK*. Je redimensionne également *Annuler* qui est un peu trop allongé. Je groupe ensuite les deux boutons et j'obtiens :

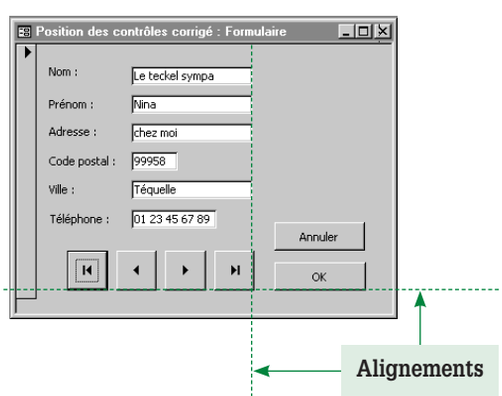


5. C'est presque fini : je n'ai plus qu'à supprimer la barre d'état et à repositionner mes trois groupes (les quatre boutons de déplacement, les zones de texte et les boutons *Annuler* et *OK*).

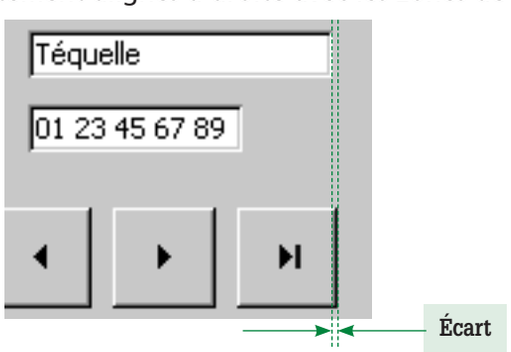


Vous remarquerez que les différents groupes sont alignés les uns vis-à-vis des autres : les boutons de déplacement sont alignés à droite avec les zones de texte et les deux boutons de validation sont alignés en bas avec ceux de déplacement. Mon idée, en faisant ce formulaire, est de regrouper les zones sur lesquelles on clique pour éviter d'avoir à trop déplacer la souris.

Voyez l'illustration qui suit.



Si vous avez l'œil, vous aurez remarqué que les boutons de déplacement ne sont pas exactement alignés à droite avec les zones de texte. Zoomons :



Cet écart est volontaire : le « rebord » grisé du bouton induit un effet d'optique. Si vous alignez exactement les deux contrôles, vous aurez l'impression visuelle d'un petit décalage.

Finalement, rendre le formulaire agréable n'est pas plus long que de mettre les contrôles en vrac. N'oubliez pas d'utiliser les sympathiques outils du menu *Format*.

5. Ergonomie du formulaire : aspect automatisation

5A. Introduction

Le formulaire peut réaliser trois types d'actions que nous allons étudier :

- l'action évidente demandée par l'utilisateur par un clic sur un menu ou un bouton de commande ;
- les actions de vérification, de remplissage automatique et d'aide à la saisie ;
- des actions plus fines, traduisant l'adaptation du formulaire à l'environnement (données saisies, traitement en cours…).

5B. L'action demandée par l'utilisateur

5B1. Principe

Ce sont donc les actions généralement déclenchées par un clic souris, événement le plus courant causé par l'utilisateur. D'ailleurs, les contrôles sur lesquels l'utilisateur clique sont généralement les boutons ou les menus.

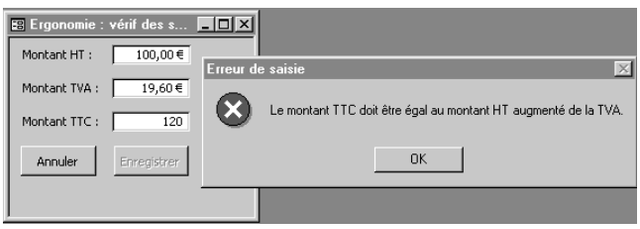
Dans l'absolu, les contrôles (boutons, menus) devraient se suffire à eux-mêmes sans plus d'explication. Dans la pratique, une notice est bien entendu nécessaire mais vous ne devez pas vous retrancher derrière elle pour ne plus faire attention aux actions proposées.
La règle est assez banale : les actions et les boutons associés doivent être explicites et tout à fait distincts. Dans le formulaire précédent, nous avions des boutons *Abandonner* et *Annuler*. Les deux mots sont sémantiquement trop proches (d'ailleurs, il apparaissait que ces deux boutons déclenchaient le même traitement).
Pour valider ou non les modifications d'un formulaire, utilisez les mots standardisés *Annuler* et *OK*. Dans les boîtes de dialogue, utilisez des boutons *Oui* et *Non* quand vous posez des questions.
D'une façon générale, un formulaire doit posséder quelques boutons (2 ou 3) de commande et pas plus. Vous comprenez bien qu'un formulaire avec 20 boutons est une télécommande. On se perd dans les différents choix. Dans ce cas, il est vraisemblable que le formulaire pourrait être scindé en plusieurs ; sinon, il faut employer une barre de menus.

De plus, vous devez toujours laisser une porte de sortie neutre (ne faisant rien) à l'utilisateur. C'est en général le bouton *Annuler* qui ferme le formulaire sans prendre en compte les modifications éventuellement faites. C'est important pour plusieurs raisons :

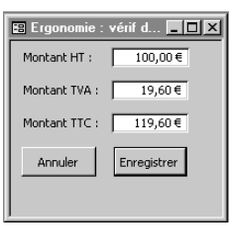
- l'utilisateur peut avoir ouvert le formulaire par erreur, ou juste pour voir ce qu'il contenait. Il veut alors le fermer en étant sûr de n'avoir rien « cassé » ;
- l'utilisateur voulait utiliser ce formulaire, mais il n'arrive pas à y rentrer des valeurs correctes (quelle que soit la raison : donnée fausse, incompréhension du formulaire…). Il faut donc pouvoir fermer le formulaire sans que les données incorrectes soient enregistrées.

Nous allons donc rajouter un bouton *Annuler* au formulaire de saisie des montants HT et TTC fait dans le paragraphe 5C3 (finalisons !) de la séquence précédente. (Bien entendu, il faudrait du code derrière ce bouton mais notre propos n'est pas là.)

Saisie incorrecte :



Saisie correcte :



Dans tous les cas, le bouton *Annuler* reste accessible.
N'hésitez pas à étudier les différents formulaires des applications Windows pour vous forger un style !

5B2. Prévoir les actions interdites

Nous avons vu dans la séquence 12 la désactivation des boutons de déplacement lorsque l'on ne devait pas s'en servir. C'est assez pénible à faire : pour chaque formulaire, il faut gérer l'activation ou la désactivation de chaque contrôle en fonction des différentes situations possibles, qui peuvent être nombreuses.
Nous allons travailler sur un exemple classique : la navigation dans les enregistrements. Lorsque vous êtes sur le premier auteur de la base de données, cela n'a pas de sens de cliquer sur l'auteur précédent. De même, quand je suis sur le dernier auteur, il n'y a pas d'enregistrement suivant. L'idée est toujours que dans une situation donnée, certaines actions deviennent inadéquates et ne doivent pas être faites.

Et là, quatre cas possibles, du pire au meilleur :

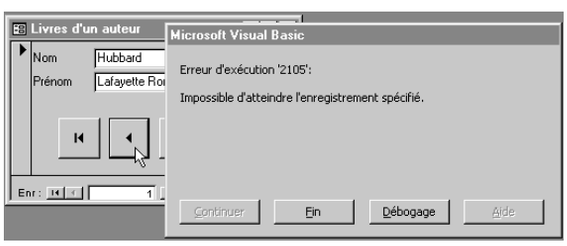
1. Lorsque l'on réalise l'action interdite, votre application plante lamentablement (c-à-d. s'arrête brutalement avec un message d'erreur Windows). C'est le pire qui puisse arriver : votre application est mal conçue. Vous n'êtes pas un professionnel❶.
2. Lorsque l'on réalise l'action interdite, on arrive dans un état inconsistant. Par exemple, dans la séquence 12 (paragraphe 4C), lorsque l'on affichait les livres d'un auteur puis que l'on changeait d'auteur, l'ancienne liste de livres restait affichée, ce qui était incohérent. Deux solutions pour éviter cela : empêcher de changer d'auteur (ce que l'on avait fait) ou mettre la liste à jour lorsque l'auteur change.
3. Lorsque l'on réalise l'action interdite, vous avez un message vous avertissant que c'est impossible. C'est mieux car l'application ne plante pas, mais la démarche est assez curieuse : vous laisser faire une erreur pour vous annoncer « non, on ne peut pas. ».

❶ *Bien entendu, j'ajoute immédiatement un bémol. Si toutes les actions interdites (soit plusieurs par formulaire) entraînent un plantage, vous avez fait un travail lamentable. Imaginez que dans votre voiture, si vous tiriez trop le frein à main il s'arrache, si vous appuyez trop sur les pédales, elles se bloquent… c'est invraisemblable !*
En revanche, si votre application ne plante que dans quelques situations particulières, on arrive sur le terrain du bug. On peut éventuellement vous reprocher de ne pas avoir suffisamment testé votre application, mais bon, une grosse application complexe sans bug, c'est comme un rapport de 1 000 pages sans coquille… le rêve de tout concepteur ou rédacteur.

4. On ne peut pas réaliser l'application interdite (bouton désactivé). C'est intuitif et cela permet à l'utilisateur de bien comprendre la logique de l'application puisqu'il voit les actions possibles ou non au fur et à mesure du déroulement de l'application.

Comme les outils de développements visuels (VB, Delphi…) permettent la désactivation des contrôles, il faut s'en servir !
Nous allons illustrer ces différents cas avec un exemple concret : le formulaire de la navigation dans la table des auteurs (vu séquence 12, paragraphe 4A). J'ai ajouté la barre d'état du formulaire pour que vous voyiez l'enregistrement sur lequel on est. Nous allons à chaque fois faire une action interdite : positionné sur le premier auteur, je veux accéder à l'auteur précédent (qui n'existe pas).

Premier cas : quand on fait une action interdite, l'application plante.

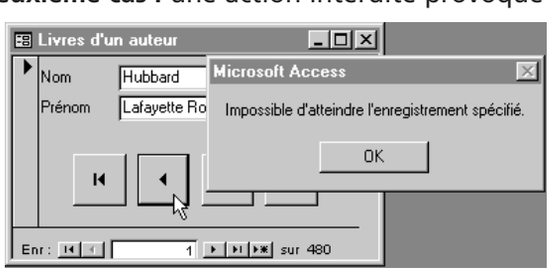


(Pour faire cela, je suis allé dans le code VB associé à l'événement Sur clic *du bouton et j'ai mis en commentaire la gestion des erreurs (instruction* On error goto*).)*

Là, c'est pire que tout :

- si l'utilisateur clique sur *Débogage* (action par défaut), il accède au code VBA donc à des choses dont il ne soupçonnait même pas l'existence. Et alors, que va-t-il faire ?
- s'il clique sur *Fin*, il retrouvera l'application (car *Fin*, c'est sous-entendu du code VB). Mais comment oser cliquer sur *Fin* ? Pour l'utilisateur, cela veut plutôt dire que tout va se terminer !

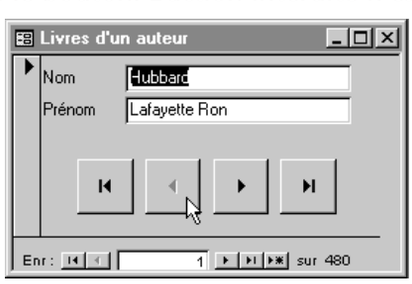
Deuxième cas : une action interdite provoque un message d'erreur.



(Pour cela, j'enlève le commentaire sur la ligne de gestion de l'erreur.)

Là, tout va bien : on a un message explicite ; la seul façon d'en sortir, c'est de cliquer sur *OK*. Vous devez donc accuser réception du message d'erreur.

Troisième cas : on désactive le bouton. L'action interdite n'est donc plus possible.



Je peux cliquer tant que je veux, il ne se passe rien. Ce n'est pas traumatisant pour l'utilisateur, puisque l'aspect grisé du bouton indique bien quelque chose (à vous d'expliquer dans votre notice que les contrôles grisés sont désactivés car leur action est incohérente à ce moment).

Notez que nous ne savons pas faire cela pour le moment (le code VB généré dans la séquence précédente n'a su mettre en œuvre que la gestion d'erreur, ce qui n'est déjà pas si mal). Pour désactiver ce bouton lorsque l'on arrive sur le premier enregistrement, il faut pouvoir accéder à la source de données par programmation afin de tester si l'on se trouve sur le premier enregistrement ou non. Cela fera partie du cours de l'année prochaine…
Il y a une alternative à tout cela : on pourrait laisser le bouton tel qu'il est (sans le désactiver) et, par code, tester si l'on doit faire quelque chose ou non. La procédure événementielle liée au clic sur le bouton serait alors quelque chose de ce style :

si enregistrement courant <> premier enregistrement
  alors
    se positionner sur l'enregistrement précédent
fin-si

(Notez qu'il n'y a pas de branche sinon*… il ne se passe donc rien si l'on est sur le premier enregistrement.)*

Cette solution est à **proscrire**. En effet, c'est incompréhensible pour l'utilisateur : dans certains cas, cliquer sur le bouton fait quelque chose, dans d'autres cas il ne se passe rien. Ce n'est pas acceptable.

Certains n'emploient pas la propriété *Enabled* (gérant l'(in)activité) mais *Visible* (qui affiche ou non le contrôle). Ainsi, au grès du fonctionnement, vous voyez des contrôles apparaître et disparaître. Je déconseille cette solution pour plusieurs raisons :

- voir des choses apparaître puis disparaître est très perturbant : on ne peut pas s'habituer au formulaire puisqu'il change constamment ;
- le fait de montrer qu'une action existe mais est inaccessible pour le moment permet d'appréhender la logique de l'application bien plus facilement qu'en effaçant le bouton correspondant (car alors, on ne sait pas que cette action existe) ;
- ensuite, cela accentue l'aspect à éviter « l'informatique vous domine ». Face à une application où les contrôles apparaissent puis disparaissent, je me sens infantilisé : je pense toujours qu'une super fonctionnalité m'est inaccessible car je n'ai pas trouvé l'action à réaliser pour faire apparaître le bouton permettant de l'utiliser.

Une preuve que j'ai raison ? Montrez-moi une application professionnelle jouant au magicien avec les contrôles !

5C. Vérification des données, remplissage et aide à la saisie

Un formulaire (Access, VB, Delphi…) reprend le concept du formulaire papier. Cela dit, l'informatique n'est pas inerte comme un vulgaire bout de papier. Vous pouvez (voire devez !) écrire du code assistant et contrôlant la saisie des utilisateurs. Nous avions déjà vu cela avec les propriétés des champs des tables. L'intérêt est le suivant :

- on évite les fautes de frappe rendant les données inconsistantes (entrer 1870 au lieu de 1970 comme année de naissance) ;
- on aide la saisie en proposant un masque : le gabarit « __/__/____ » laisse penser que l'on attend une date ;
- on peut vérifier les données saisies en s'assurant par exemple que le montant TTC est égal au montant HT plus la TVA.

Le but est double :

- améliorer la saisie des données (convivialité de l'application, souci de l'utilisateur) ;
- améliorer la qualité des données (efficacité de l'application).

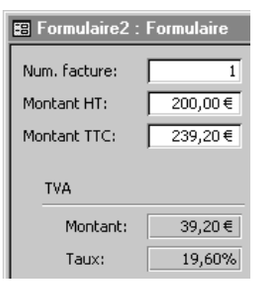
Nous avions déjà abordé ces points dans les séquences précédentes (séquences 9 et 12). Il est néanmoins utile de reformuler cela dans le cadre général de l'ergonomie.
Nous allons voir différents exemples.

5C1. Vérification des données

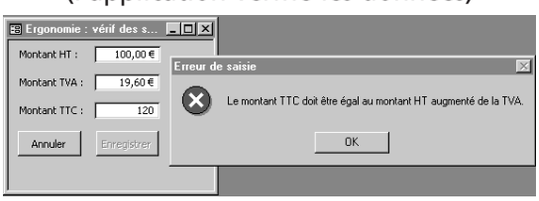
La vérification des données peut être faite par l'application ou par l'utilisateur. Prenons l'exemple de la saisie d'un montant TTC et HT. Deux solutions s'offrent à vous :

- vous affichez le montant de TVA et laissez l'utilisateur vérifier si ce montant est correct ;
- l'utilisateur doit également saisir la TVA. L'application vérifie la cohérence des trois nombres et n'accepte la saisie qui si tout va bien.

affichage du montant de TVA
(l'utilisateur vérifie les données)



saisie de la TVA
(l'application vérifie les données)



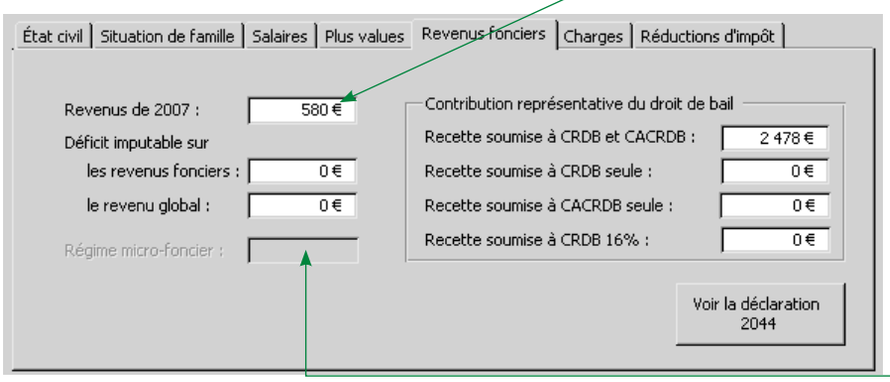
Nous avions vu cela dans les séquences 9 et 12 (et ci-dessus). Je vous rappelle que la seconde version est préférable.

5C2. Aide à la saisie

L'idée est d'automatiser tant que l'on peut le formulaire pour limiter au maximum les saisies de l'utilisateur. En effet, après ma règle disant de ne jamais lui faire confiance, j'en rajoute en disant que chacune de ses interventions est un problème potentiel. Bref, moins il interagit avec l'application, mieux elle se porte.

Reprenons notre formulaire de déclaration de revenus en détaillant le régime d'imposition de revenus fonciers❶ :

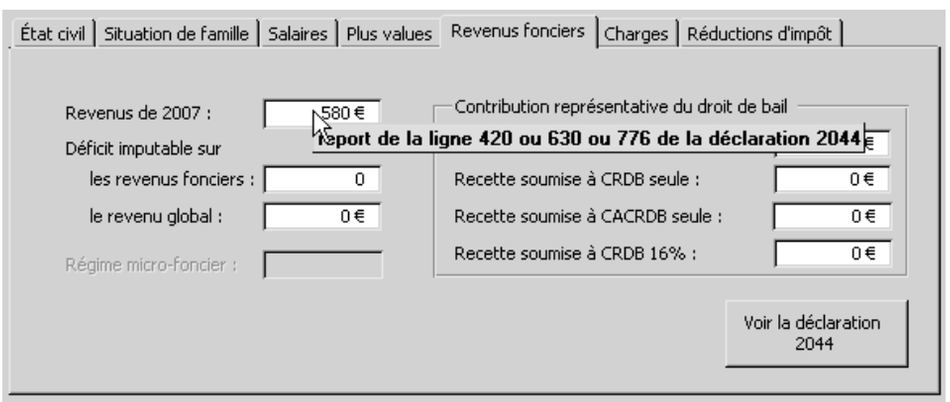
Si vous avez perçu plus de 15 000 € de loyers ou ne voulez pas du régime micro-foncier, vous devez remplir une déclaration 2044 où vous détaillerez toutes les recettes locatives et les charges. La déclaration 2044 vous guide dans les différents calculs jusqu'à obtenir le revenu foncier imposable (grossièrement : loyers – charges diverses dont une déduction forfaitaire de 14 %). Ce revenu doit être reporté **ici**. *C'est beaucoup plus lourd que le régime micro-foncier, mais plus avantageux si vous avez beaucoup de charges (notamment un emprunt).*



Si vous avez encaissé moins de 15 000 € de loyers, vous pouvez opter pour le régime micro-foncier. L'intérêt ? Vous bénéficiez d'un abattement forfaitaire de 40 % correspondant aux charges diverses que vous avez pu supporter. Vous n'avez alors qu'à indiquer le montant des loyers perçus dans la case désactivée **ici**.

On a donc le choix entre deux régimes distincts et exclusifs : soit l'abattement forfaitaire de 40 %, soit l'abattement de 14 % plus charges réelles.
L'important est que c'est soit l'un, soit l'autre. Si je rentre mes loyers dans la case *revenus de 2007*, je ne dois pas pouvoir les rentrer dans le régime micro-foncier… et inversement. C'est pourquoi une seule des deux zones est active à un moment donné.
Je vous rappelle que la déclaration suivant le régime normal (14 % plus charges) oblige à la déclaration 2044 détaillant tous les chiffres. Les chiffres indiqués dans le formulaire ci-dessus sont donc un simple report des chiffres inscrit dans la déclaration 2044. Mais quels chiffres reporter ?
Une solution serait d'utiliser la notion d'info-bulle (texte d'explication affiché au niveau du curseur lorsque qu'il est sur un contrôle).

Voici un exemple.



❶ *Ce cours n'est pas à jour des dernières lois fiscales.*

Ainsi, dès que vous mettez votre curseur sur la zone des revenus, on vous explique où aller chercher le chiffre. Certes, on aurait pu mettre cette information avec le libellé de la zone (c'est ce qui est fait dans la déclaration papier). Mais cela aurait énormément alourdi le formulaire avec des informations inutiles sauf au moment de remplir une case. L'info-bulle est donc parfaitement indiquée.
Comment les mettre en œuvre ? C'est trivial : Il suffit de mettre le texte dans la propriété *Texte d'info-bulle* du contrôle concerné :

Cela dit, dans ce cas précis, l'info-bulle n'est pas la meilleure solution : cette technique montre le manque d'inventivité du programmeur (moi) puisqu'il se contente d'émuler le formulaire papier avec les outils informatiques (les notes deviennent des info-bulles).

La solution la plus efficace est d'exploiter pleinement l'automatisation permise par le formulaire. Au lieu de dire quel chiffre reporter, l'application le reporte pour nous.
Dans ce cas, il faut programmer le formulaire 2044 pour qu'à sa fermeture, il aille écrire les informations utiles dans le formulaire 2042. On va donc utiliser l'événement *Sur fermeture* du formulaire 2044 pour reporter les valeurs dans le formulaire 2042. Si les deux formulaires s'appellent *Form2042* et *Form2044*, on aura du code de ce style pour l'événement *Sur fermeture* de *Form2044 :*

```
Private Sub Form_Close()
  With Forms![Form2042]
    .Revenus2007.Value = RevenuFoncierNet.Value
    .CACRDB.Value = CACRDB.Value
    .CRDB.Value = CRDB.Value
    ...
  End With
End Sub
```

Dans les affectations :

- les variables de gauche sont les contrôles du formulaire *Form2042* ;
- celles de droite sont les contrôles du formulaire courant (*Form2044*).

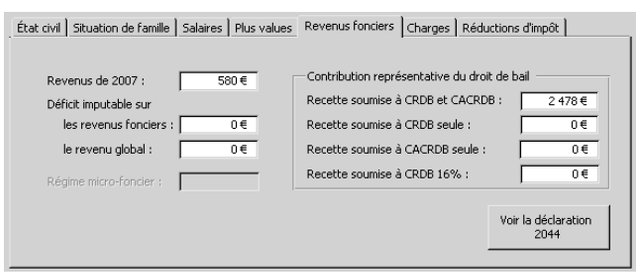
Je vous rappelle que le *With* est une factorisation. Sans lui, on aurait écrit ce code plus lourd :

```
Private Sub Form_Close()
  Forms![Form2042].Revenus2007.Value = RevenuFoncierNet.Value
  Forms![Form2042].CACRDB.Value = CACRDB.Value
  Forms![Form2042].CRDB.Value = CRDB.Value
    ...
End Sub
```

6. Accès aux contrôles

6A. Introduction

Reprenons notre formulaire de déclaration :



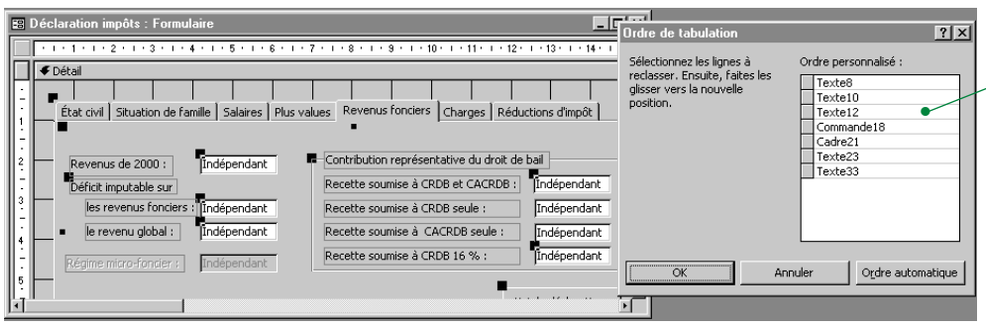
Supposons que nous n'ayons pas de remplissage automatique… Nous devons donc rentrer 7 chiffres (zones actives). Souvenez-vous du formulaire permettant de rentrer un livre (séquence 7) : il contenait plus de 15 zones de saisie !

Comment passer de l'une à l'autre ? Une solution consiste à cliquer dans la zone à atteindre. Mais c'est fastidieux : vous devez taper au clavier la valeur, lâcher le clavier de la main et des yeux pour reporter l'une sur la souris et les autres sur l'écran, cliquer sur la nouvelle zone puis reporter toute votre attention sur le clavier… c'est lent et désagréable.
Il y a deux façons d'accéder aux différents contrôles des formulaires : la tabulation ou le raccourci clavier.

6B. La tabulation

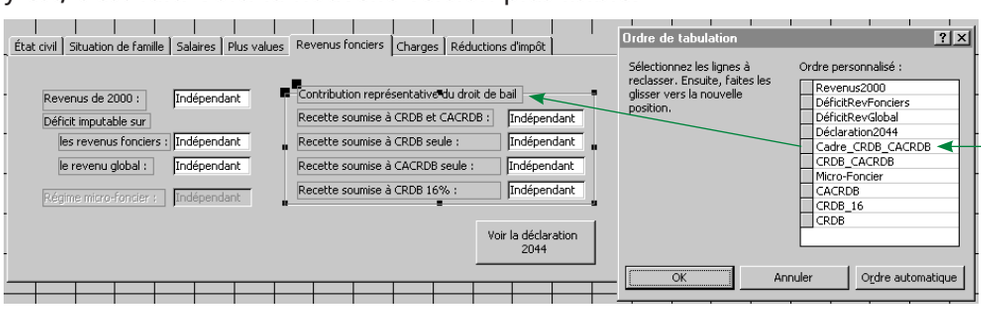
Elle permet un parcours complet des contrôles. L'idée est simple : vous définissez l'ordre dans lequel les contrôles doivent être parcourus (généralement de haut en bas ou de gauche à droite). Quand vous êtes sur un contrôle, appuyer sur la touche *Tab* (tabulation) vous fait passer sur le contrôle qui suit dans la liste. Quand vous êtes sur le dernier et faites *Tab*, vous revenez au premier.
Il est possible d'exclure des contrôles de la liste de tabulation. Dans ce cas, la tabulation ne permet pas d'y passer. Il faut alors cliquer dessus. C'est utile lorsque le contrôle est utilisé dans une saisie sur dix : il vaut mieux cliquer une fois sur dix pour y accéder qu'appuyer neuf fois sur dix sur *Tab* pour le sauter.
Pour définir l'ordre, c'est très simple : lorsque le formulaire est en mode *Création*, menu *Affichage/Ordre de tabulation…* et vous obtenez une fenêtre de paramétrage.

Dans l'exemple de notre formulaire de déclaration :



Je vais mettre les contrôles dans l'ordre qui me convient.

Je n'ai pas joué le jeu en ne nommant pas correctement mes contrôles. Et là, je suis coincé ! Qui est *Texte10*, *Texte12*… ? D'ailleurs, il me manque des contrôles, je ne les ai pas tous sélectionnés.
Bon, je nomme (propriété *Nom*) correctement mes contrôles et je reviens.
Ca y est, c'est fait. Voici la nouvelle version plus lisible.



Vous remarquerez ce contrôle. C'est un groupe d'options qui ne contient aucun bouton d'option. Il me permet juste de regrouper visuellement des zones de texte. Sa seule fonction est esthétique (ergonomique).

Le principe est le suivant : on place tous les contrôles dans l'ordre souhaité. Si l'on ne veut pas d'un contrôle, on doit le mettre quand même, mais on mettra sa propriété *Arrêt tabulation* à *Non*. (Par défaut, cette propriété vaut *Oui*). L'intérêt de cette nouvelle propriété ? Et bien, on pourra rendre un contrôle accessible ou non par la touche *Tab* par programmation, en changeant la valeur d'*Arrêt tabulation*.

Notez que cette fenêtre est en fait un assistant. Vous pouvez faire le travail vous-même en renseignant la propriété *Index tabulation* des contrôles : *1* pour le premier, *2* pour le deuxième… Bien entendu, cette fenêtre est plus pratique.

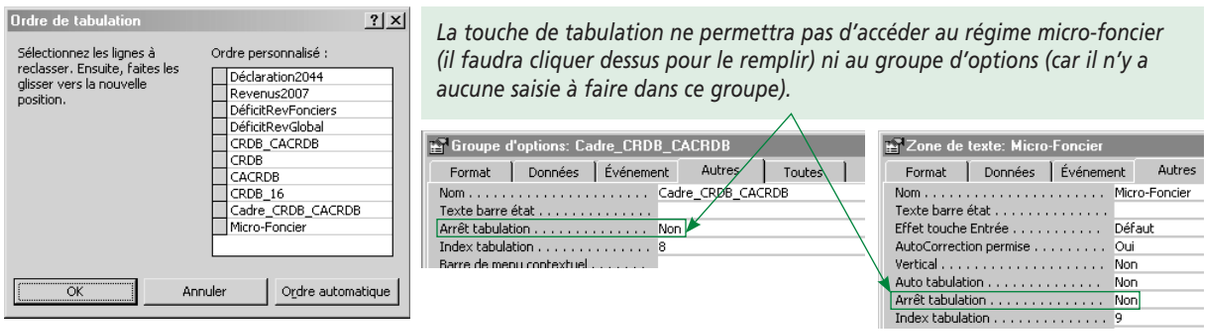
Ici, nous ne définissons aucun ordre sophistiqué. Je vous propose de mettre en premier le bouton de la déclaration 2044 (car cette déclaration 2044 me permet d'obtenir les chiffres permettant de remplir ce formulaire❶), puis les contrôles de haut en bas et de gauche à droite.

Comme j'estime que le régime micro-foncier ne sert que rarement, je ne veux pas m'arrêter dans la zone de texte correspondante avec la tabulation. Je place donc ce contrôle n'importe où puisqu'il compte pour du beurre (en pratique, je le mets plutôt en premier ou en dernier) et je change sa propriété *Arrêt tabulation* à *Non*.

❶ *Je vous rappelle mon hypothèse pour ce paragraphe : le remplissage des zones de texte n'est pas fait automatiquement. C'est à vous de reporter les chiffres.*

De même, le contrôle groupe d'options ne contient aucun contrôle (sous-entendu aucun bouton d'option). Il ne stockera (n'aura) donc aucune valeur. Ainsi, lui attribuer le *focus* (le rendre actif donc prêt à la saisie) fait disparaître ce dernier puisqu'il ne possède aucun contrôle permettant cette saisie ! Il faut donc également l'exclure de la liste.

Au final, on obtient :



D'une façon générale, l'ordre de tabulation doit suivre l'ordre naturel de lecture du formulaire. Imaginez que *Tab* fasse voler le focus aux extrêmes du formulaire… Il faudrait plusieurs secondes pour trouver quel est le contrôle courant.

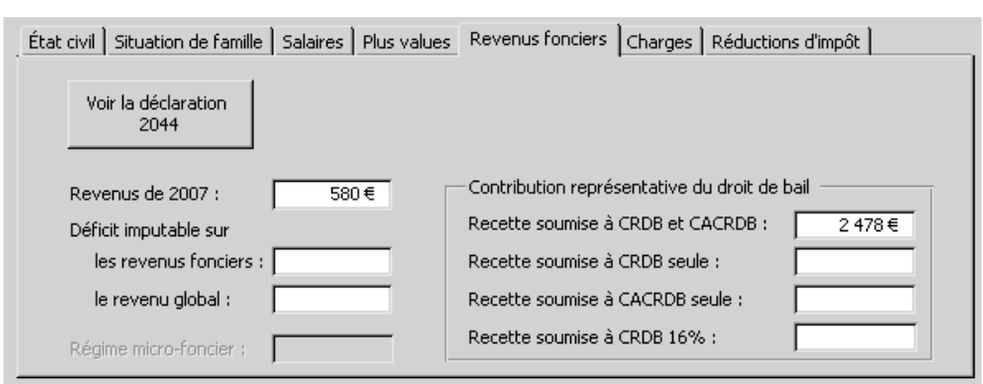
En fait, si l'ordre de remplissage défini par les tabulations ne correspond pas à l'ordre de lecture naturel (intuitif) du formulaire, ce dernier est mal conçu : vous devrez réorganiser les contrôles.

Ici, j'active le bouton en bas à droite, puis la touche tabulation me fait remonter en haut à gauche… Ce n'est pas raisonnable. Il me faut synchroniser la position physique du contrôle par rapport à son ordre dans la liste de tabulation.

J'ai deux solutions pour remédier au problème :

- je place le bouton en dernier dans ma liste de tabulation ;
- je déplace le bouton pour le mettre en début de formulaire.

Comme il est logique de remplir la déclaration 2044 pour ensuite reporter les chiffres dans le formulaire courant, je vais déplacer le bouton de commande. Mon formulaire est finalement le suivant :



En conclusion, je dirai qu'il faut bien réfléchir pour concevoir un formulaire ! On les réalise rarement d'un jet. On travaille plutôt par étapes jusqu'à obtenir une ergonomie satisfaisante.

6C. Raccourcis claviers

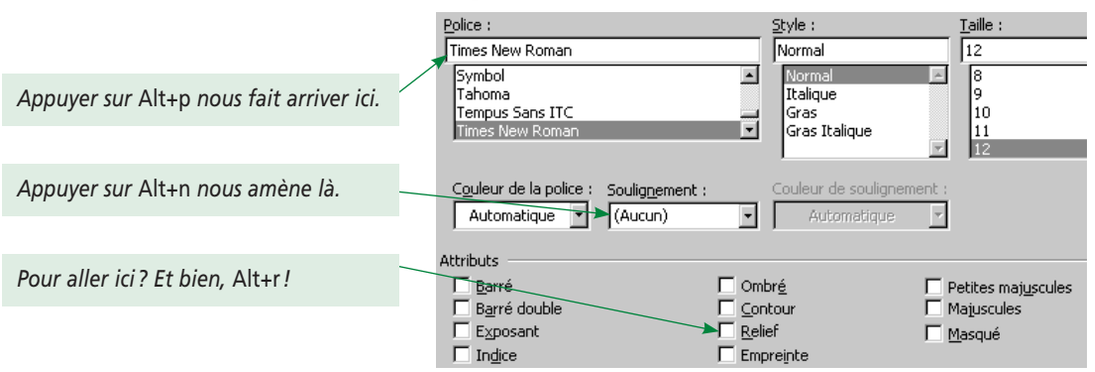
6C1. Explicites

Il y a finalement trois façons pour accéder à un contrôle :

- cliquer dessus ;
- appuyer sur la touche de tabulation jusqu'à y accéder ;
- utiliser un raccourci clavier.

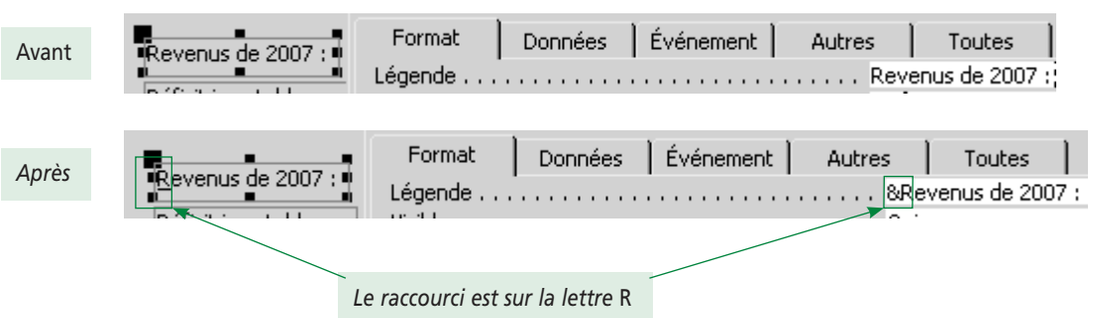
Le raccourci clavier et le clic permettent d'accéder directement au contrôle (alors qu'il faudrait éventuellement plusieurs appuis sur la touche *Tab* pour y arriver). L'intérêt du raccourci clavier est sa rapidité : il est beaucoup moins long d'appuyer sur une touche que de prendre en main la souris et de cliquer sur un contrôle.
Cela dit, les raccourcis clavier ne sont pas très conviviaux. Ils étaient cruciaux aux origines des environnements graphiques où tout le monde n'avait pas de souris❶. Maintenant, on ne s'en sert plus guère.
Comment mettre en œuvre un raccourci clavier ? Visuellement, le libellé du contrôle possèdera un caractère souligné. En appuyant sur la touche *Alt* plus cette lettre, vous accédez directement au contrôle.

Illustrons cela avec une boîte de dialogue de Word :



Comment créer un raccourci clavier ? C'est tout simple !

Dans la propriété *Légende* du libellé du contrôle, mettez un caractère *&* avant la lettre servant de support au raccourci. Ce *&* sera converti en soulignement pour l'affichage du libellé. Illustrons :



❶ *Et oui, cette époque a existé… Cela date d'avant Windows 95, soit 1995.*

La mise en œuvre du raccourci clavier est très simple. Pour autant, cela requiert beaucoup de rigueur :

- si l'on emploie les raccourcis clavier dans un formulaire, chaque contrôle doit en être équipé, sinon l'on est incohérent❶ ;
- c'est une évidence (mais pas si évident à faire), lorsque j'appuie sur *Alt+p* (par exemple), je ne dois arriver que sur un seul contrôle ! Bref, il faut faire attention à ne pas utiliser deux fois le même raccourci clavier dans un formulaire. En pratique, on utilise la première lettre du libellé. En cas de doublon, on choisit une autre lettre (si possible intuitive) pour l'un des contrôles.

Comme je l'ai dit précédemment, les raccourcis clavier ne sont plus très employés maintenant que la souris est généralisée. De plus, nous venons de voir qu'il n'était pas évident de les définir sans erreur.

La règle sera donc au choix :

- les employer correctement dans toute l'application ;
- ne pas les employer du tout.

Je vous conseille de **ne pas** les employer : l'intérêt est plus qu'anecdotique et le risque d'erreur est grand. L'ultime argument ? Vu les cours que j'ai rédigé pour vous, je me suis beaucoup promené dans les écrans des outils Microsoft. Et bien, j'y ai trouvé des fautes de français et des erreurs d'ergonomie. Voici notamment sous Word 2000 une partie des fenêtres des menus *Format/Police…* et *Format/Paragraphe…*

Observez-les… qu'en pensez–vous ?

Les onglets de la fenêtre *Police* n'ont pas de raccourcis clavier…❷

6C2. Implicites

Il y a deux raccourcis claviers qu'il faut en revanche toujours employer. Ils sont implicites, à savoir qu'aucun élément visuel ne les indique. Pourquoi ? Parce qu'ils sont évidents…
Presque tout formulaire possède les deux boutons *OK* et *Annuler*, qui permettent de fermer la fenêtre en prenant en compte (bouton *OK*) ou non (bouton *Annuler*) les modifications faites.
Les règles d'ergonomie veulent que deux raccourcis claviers existent pour quitter une fenêtre (boîte de dialogue, formulaire…) :

- pour quitter une fenêtre en validant, on appuie sur *Entrée* (*Enter*) ;
- pour quitter une fenêtre en annulant, on appuie sur *Échap* (*Escape*).

Bien entendu, il est impossible de matérialiser ces raccourcis avec un soulignement puisqu'ils n'utilisent pas la touche *Alt* ! C'est pourquoi ils demeurent implicites.

❶ *Erreur classique et qui fait très mal en pratique des techniques informatiques sur la compétence maquettage.*

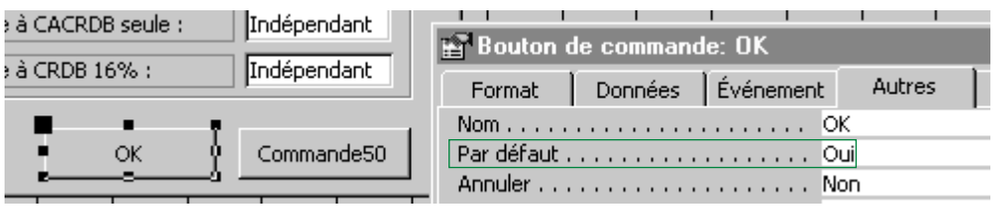
❷ *Erreur corrigée dans Word 2003.*

Comment les définir ? C'est l'objet des deux propriétés *Par défaut* et *Annuler* :

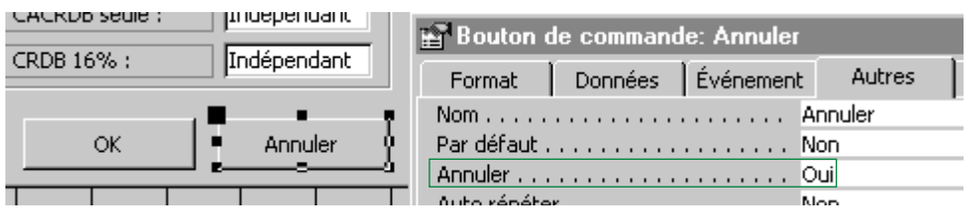
- un bouton dont la valeur de la propriété *Par défaut* est *Oui* déclenchera le code lié à son événement *Sur clic* lorsque l'on appuiera sur *Entrée* ;
- un bouton dont la valeur de la propriété *Annuler* est *Oui* déclenchera le code lié à son événement *Sur clic* lorsque l'on appuie sur *Échap*.

Bien entendu, il ne peut y avoir dans un formulaire qu'un bouton répondant à *Échap* et un répondant à *Entrée*. Il est donc impossible d'attribuer *Oui* aux propriétés *Par défaut* ou *Annuler* de plusieurs boutons.
Nous allons rajouter à notre formulaire de déclaration les deux boutons de validation et d'annulation :

Le bouton *OK* (vous remarquerez qu'à côté, j'ai placé le futur bouton *Annuler* pas encore défini) :



Le bouton *Annuler* :



Bien entendu, le bouton d'affichage de la déclaration 2044 n'est ni un bouton de validation, ni un bouton d'annulation :

7. Terminons sur trois choses rapides

7A. L'état

J'ai écrit plus de vingt pages sur l'ergonomie du formulaire… qu'en sera-t-il pour celle de l'état ? Ma foi, ce sera vite vu : l'état est un mélange de document texte (Word) et de formulaire :

- toutes les règles de présentation associées au traitement de texte doivent être respectées (voir le fascicule de présentation de la formation) ;
- les règles vues pour les formulaires sont toujours d'actualité, notamment l'alignement des contrôles.

7B. Le compactage d'une base de données

Lorsque vous travaillez dans votre base de données, Access stocke tous les éléments créés dans le fichier *MDB*. En pratique, le fichier s'allonge toujours : si un nouvel élément le fait grandir, la suppression d'un élément ne le fait pas rétrécir.
En fait, si après une journée de travail sur des formulaires très complexes vous supprimez ces derniers, votre base restera énorme, même si elle ne contient aucune donnée. Il n'est pas rare qu'une base passe de 400 ko à 1,5 Mo au cours d'une session de travail.

Cela pose plusieurs problèmes :

- d'une part, plus la base est grosse, plus elle prend de place❶ (notamment sur une disquette si vous voulez la transporter ou la sauvegarder) ;
- d'autre part une base grosse sans raison (contenant des objets supprimés) s'exécutera moins rapidement pour des raisons techniques que je ne développe pas ;
- tant que l'on ne compacte pas la base, le code VBA est interprété et non compilé… bref, il est plus lent.

Il est donc important de compacter régulièrement votre base et au moins lorsqu'elle est finie avant de la livrer à l'utilisateur.
Pour le faire, c'est très simple : *Outils/Utilitaires de base de données/Compacter une base de données*. Et c'est tout (allez prendre des informations dans l'aide). Cela prend juste quelques secondes !

7C. À vous de voir…

Ce dont je ne parle pas, mais que vous pouvez étudier vous-même avec l'aide :

- les fichiers MDE (*Outils/Utilitaires de base de données…*) ;
- la gestion des droits d'accès (*Outils/Sécurité*) ;
- la gestion des réplicas (*Outils/Réplication…*). Cela permet d'avoir votre base sur un ordinateur, une copie sur un autre (classiquement un portable) et de les synchroniser ;
- l'ergonomie de l'application globale (*Outils/Démarrage…*) : accès aux touches de accourci d'Access, quel formulaire afficher au démarrage. C'est sans doute le plus intéressant pour vous.

Tout cela fait partie du cadre très large de l'ergonomie !

❶ *Oui, je sais, c'est un peu évident.*

Synthèse

L'ergonomie est souvent le parent pauvre chez les informaticiens débutants (qui n'ont jamais rencontré d'utilisateur). Pourtant, c'est quelque chose d'indispensable car :

- le minimum du professionnalisme est de fournir quelque chose de bien fait, aspect visuel y compris ;
- hormis les bugs plantant l'ordinateur ou des problèmes de lenteur importants (plusieurs secondes de délai entre une action et sa réalisation), la seule chose que l'utilisateur perçoit de l'application, c'est l'interface. Son évaluation de votre travail se basera à 90 % sur elle (sous-entendu que les fonctionnalités commandées sont présentes). Oui, je sais, c'est injuste pour votre talent de programmeur. Mais bon… quand on achète une voiture on s'intéresse souvent beaucoup plus à son interface qu'à ses qualités intrinsèques, non ?

Les règles d'ergonomie tiennent en quelques phrases choc :

- ne surchargez pas un formulaire. N'hésitez pas à le structurer (sous-formulaires, onglets, autre formulaire ouvert par un bouton) ;
- soyez très circonspect dans l'usage des couleurs et des mises en forme ;
- employez les contrôles corrects en fonction du type de valeur à saisir ou à afficher ;
- alignez les contrôles les uns vis-à-vis des autres ;
- vérifier la cohérence des actions proposées par le formulaire ;
- aidez la saisie : désactivez les commandes ou les zones qui ne doivent pas être accessibles, employez les info-bulles et définissez l'ordre de tabulation ;
- vérifiez à chaque fois que c'est possible la validité des données saisies (une date de facture ne doit pas être postérieure à la date du jour). La vérification se fera dès la saisie et non une fois tout le formulaire rempli ;
- évitez d'employer les raccourcis clavier (sauf *Entrée* pour *OK* et *Échap* pour *Annuler*) ou alors, faites-le avec sérieux !
- lorsque vous affichez des messages (avertissement, question, erreur), soyez concis : personne ne lira un message de dix lignes ;
- lorsque l'utilisateur lance une commande dangereuse (suppression de données), demandez systématiquement confirmation❶ ;
- dans le doute, faites simple et sobre ;
- dans tous les cas, soyez cohérent et n'oubliez pas que vous ne faites pas l'interface pour vous, mais pour les autres.

Bien entendu, l'ergonomie ne se résume pas à cela. Il existe des livres imposants de règles très précises et pointilleuses. Mais bon, ce sont des règles utiles pour des logiciels professionnels très complexes et sans objet pour nous.

Une dernière chose. Ces concepts d'ergonomie s'appliquent à tout développement visuel, soit Access, mais aussi VB, Delphi…

❶ *Sous Dos, on supprime un fichier avec la commande* **del**. *Il est alors immédiatement supprimé sans espoir de récupération. Depuis Windows, la suppression vous demande confirmation avant de… déplacer le fichier dans la corbeille où vous pouvez toujours le récupérer. Lorsque vous videz la corbeille, Windows vous demande à nouveau confirmation. Bref, il faut être très fatigué pour supprimer un fichier important !*

Devoir auto-corrigé

Fiches de compte-rendu du tutorat

Durée approximative : 1/2 h pour lire le sujet, 3 h pour le réaliser.

Le corrigé de ce devoir vous est fourni. Il est constitué de la base de données finale et d'une dizaine de pages d'explications. Jouez le jeu et ne le consultez qu'après avoir réalisé le devoir !

1. Descriptif de l'activité

Lorsque vous vous inscrivez à la formation de BTS (première ou seconde année), le CNED vous communique les coordonnées d'un tuteur.
Le tuteur est un enseignant de BTS chargé de répondre à vos questions et de vous guider pour le stage et les actions professionnelles afin de préparer les deux épreuves orales d'informatique du BTS.
Bien entendu, le tutorat est une activité rémunérée. Mais comment ? Le CNED paye le tuteur en fonction du travail fait (ce qui est logique), donc en fonction du nombre d'étudiants suivis.
Le principe est le suivant : en début d'année, le CNED communique une liste d'étudiants dans un classeur Excel à chaque tuteur (réciproquement, tous les étudiants de la liste recevront les coordonnées de leur tuteur). Bien entendu, la nature des cours par correspondance fait que beaucoup d'étudiants démissionnent. En pratique, un tuteur va recevoir une liste d'environ 130 étudiants, sachant qu'une vingtaine d'entre eux fera appel à lui. Bien entendu, le tuteur doit être rémunéré sur le travail fait, soit le tutorat des 20 étudiants ayant fait appel à lui et non sur les 130 inscrits.
Le CNED a donc besoin de connaître la nature des contacts entre un tuteur et ses étudiants, d'une part pour la rémunération, d'autre part pour contrôler le travail de tutorat.
Pour cela, le tuteur doit remplir une fiche de compte-rendu par étudiant suivi. Cette fiche contiendra :

- les coordonnées du tuteur (nom, prénom et *code profa* – c'est son identifiant –) ;
- les coordonnées de l'étudiant (nom, prénom et numéro d'inscription) ;
- la date et la nature de chaque contact engagé avec l'étudiant ;
- comme le tuteur coordonne l'aspect pédagogique du stage (première ou seconde année), la fiche mentionne aussi les contacts éventuels avec l'entreprise d'accueil.

En fin d'année, le tuteur envoie les fiches de compte-rendu au CNED qui va pointer le travail fait et établir la rémunération correspondante.
Bien entendu, le tuteur n'enverra pas de fiche de compte-rendu vierge pour chaque étudiant n'ayant jamais fait appel à lui. Il n'y aura une fiche que si au moins un coup de téléphone a été échangé.

En fin de sujet figurent deux annexes : une liste d'étudiants et une fiche de compte-rendu. Un fichier à télécharger (à l'adresse http://www.campus-electronique.fr/bts-informatiquegestion) contient un classeur Excel avec la liste des étudiants que vous importerez dans la base Access que vous allez construire (voir la suite).

2. L'informatisation

2A. Objet de l'informatisation

Mettez-vous à la place du tuteur : après chaque coup de fil ou mail d'un étudiant, il faut chercher sa fiche de compte-rendu et la compléter. Pour l'avoir vécu, je vous avoue que c'est assez fastidieux.
Je vous propose de réaliser une petite application base de données permettant de gérer le suivi du tutorat. L'application imprimera les fiches de compte-rendu par le biais d'un état.

2B. Cahier des charges

Voici le descriptif exact (le cahier des charges) de ce que vous devez réaliser.

2B1. Principe

L'application est propre au tuteur. Elle ne permettra donc de ne gérer que les étudiants d'un tuteur donné. L'objectif est que lors de chaque coup de téléphone, il lance son application et choisisse rapidement l'étudiant. Il a alors accès à ses informations personnelles (date de naissance et différents contacts déjà engagés). Ces informations sont nécessaires pour permettre au tuteur de se remémorer l'étudiant.
Enfin, ce formulaire doit afficher les contacts réalisés avec l'entreprise d'accueil ainsi que les coordonnées de cette dernière. On doit bien entendu pouvoir renseigner ces coordonnées quand besoin est.
En fin d'année, l'application doit permettre d'imprimer les fiches de compte-rendu comme celle présentée dans l'annexe 2.

2B2. Règles de gestion

L'indicatif d'un étudiant est constitué de groupes de chiffres (trois chiffres, deux chiffres, quatre chiffres, un chiffre) séparés par des tirets ; par exemple : 782-81-0068-7.

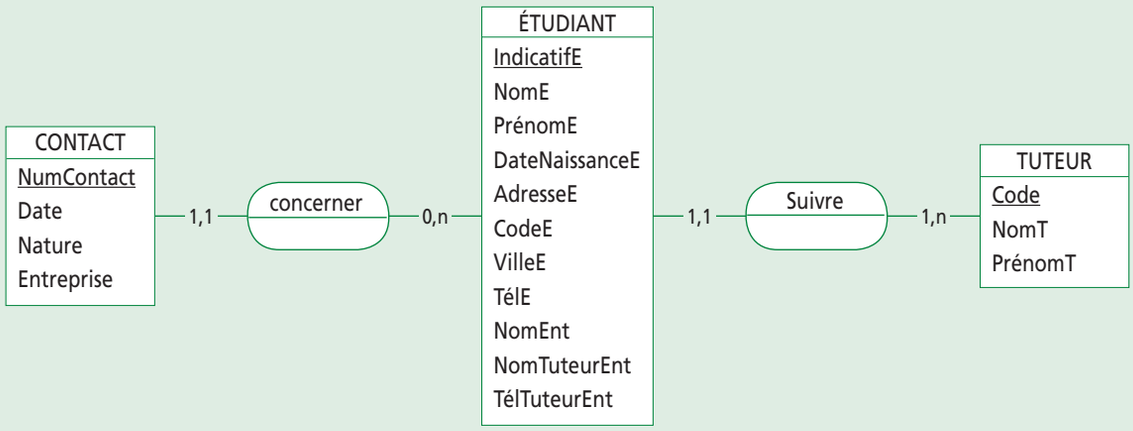
Notez que le premier groupe de chiffres identifie la classe de l'étudiant :

- 782 correspond à un étudiant inscrit en première année de BTS Informatique de Gestion ;
- 781 identifie un étudiant de seconde année.

2B3. MCD

Je vous conseille d'essayer de faire le MCD au vu du descriptif ci-dessus et de l'annexe. Il est très simple. Cela vous fera un petit exercice sympathique.

Voici une première version de la correction :



Quelques remarques :

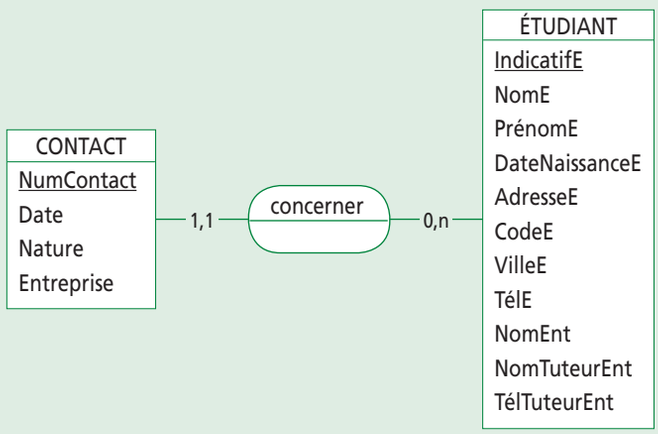
- pour éditer les fiches de compte-rendu, seuls le nom et l'indicatif de l'étudiant sont nécessaires. Cependant, le tuteur a besoin de connaître les coordonnées (adresse et téléphone) des étudiants pour pouvoir les contacter ;
- comprenez bien l'association *concerner* : pas question de faire une ternaire entre *Étudiant*, *Contact* et *Tuteur* ! En effet, quand on a l'étudiant, on a forcément le tuteur en passant par *suivre* ;
- la propriété *Entreprise* est un booléen : *vrai* si le contact est à destination de l'entreprise, *faux* si le contact est fait avec l'étudiant ;
- comme l'indicatif détermine la classe (s'il commence par *782*, c'est la première année, si c'est *781*, c'est la seconde), il est inutile d'ajouter de propriété pour connaître l'année : on la déterminera par calcul (programmation) sur l'indicatif. Cela fera l'objet d'une question à la fin du sujet.

Vous avez sans doute eu envie de matérialiser les entreprises par une entité. Mais rappelez-vous le cours d'analyse : on informatise un sujet précis et pas la terre entière ! Et là, le but de l'application est le suivi du tutorat et non la gestion des stages ! Gérer précisément les entreprises est donc inutile. Les seules caractéristiques intéressantes de l'entreprise sont :

- son nom ;
- le nom du tuteur dans l'entreprise (le *maître de stage*) ;
- le téléphone du tuteur.

Maintenant, ce n'est pas fini : le MCD est encore trop complexe. En effet, j'insiste sur le fait que la base est personnelle à chaque tuteur (moi en l'occurrence). Elle ne va donc contenir qu'un seul tuteur (moi) et uniquement les étudiants que je suis. Il est donc totalement inutile de mettre une entité *Tuteur*. C'est toujours la règle de base indiquant que lorsque l'on informatise une entreprise, on est dedans donc on ne la représente pas.

Au final, on obtient un MCD bien modeste :



2B4. Le MLD

Voici le MLD, assez simpliste :

Étudiant (IndicatifE, NomE, PrénomE, DateNaissanceE, AdresseE, CodeE, VilleE, TélE, NomEnt, NomTuteurEnt, TélTuteurEnt)

Contact (NumContact, Date, Nature, Entreprise, IndicatifE#)

Légende : clé primaire, clé étrangère#

3. Travail à faire

3A. Introduction

Il s'agit de réaliser l'application complète : tables, requêtes éventuelles, formulaires et état. Vous respecterez impérativement les contraintes suivantes :

- pour les champs des tables, vous prendrez soin de définir les propriétés (masques…) au mieux ;
- il faut définir les éventuelles relations ;
- le contenu de la table *Étudiant* viendra d'un import du fichier Excel joint (voir 3B et l'annexe 1). Le contenu de *Contact* est à votre charge par le biais du formulaire décrit ci-dessous ;
- il faudra un formulaire pour saisir les contacts avec un étudiant ou son entreprise d'accueil. En fait, je vous en demanderai deux, le premier servant d'introduction au suivant. Vous devrez réfléchir sur l'ergonomie, sachant que les formulaires doivent être remplis en temps réel (lorsque j'ai l'étudiant au bout du fil). Cela impose :
  - un choix facile de l'étudiant qui m'appelle parmi ceux dont je m'occupe ;
  - l'affichage de tous les contacts que j'ai eu avec lui ou son entreprise ;
  - pouvoir mettre à jour depuis ce formulaire les coordonnées de l'entreprise ;
  - évidemment, le formulaire doit permettre l'ajout d'un contact.
- l'état permettant d'imprimer les fiches de compte-rendu doit reprendre toutes les informations et l'ordre de présentation de celui donné en annexe.

J'insiste : l'application n'est pas trop complexe à réaliser. Ce que j'attends de vous, c'est un travail soigné et ergonomique !

3B. Importation depuis un fichier Excel

L'idée est de créer la table *Étudiant* lors de l'import des données. Dans un second temps, on définira les champs supplémentaires et les propriétés de tous les champs.
Depuis Access, menu *Fichier/Données externes/Importer* puis sélectionnez le fichier Excel (*devoir Access auto-corrigé.xls*) contenant les étudiants (*ce fichier est à télécharger voir page 242*). Attention, prenez soin de définir *Microsoft Excel* dans *Type de fichiers* sinon vous ne verrez pas les classeurs Excel.
Vous devrez ensuite étudier attentivement les écrans de l'assistant et éventuellement exploiter l'aide pour mener votre tâche à bien. Normalement, tout se fait assez naturellement.
Si vous n'y arrivez pas, inutile d'y passer trois jours : saisissez à la main une dizaine d'enregistrements.

3C. Tables et relations

J'ai dit qu'il fallait définir tout à fait correctement les champs. Je vous demande notamment un petit travail pour la clé étrangère *IndicatifE* dans *Contact* : comme un contact ne peut avoir lieu qu'avec un étudiant enregistré, j'aimerais saisir cette valeur à partir d'une liste déroulante me proposant les étudiants de la base. Et, comme les étudiants se présentent heureusement par téléphone par leur nom et pas par leur identifiant, il faudra afficher le nom mais stocker l'identifiant.
Je vous demande de faire cela dès la création du champ et non lors de la conception du formulaire : c'est la meilleure technique. Pour le faire, il suffit d'affecter au champ le type *Assistant liste de choix…* et de se laisser guider (on a fait cela quelque part dans le cours). Attention, j'aimerais une saisie la plus agréable possible : il faut que la liste affiche le nom et le prénom des étudiants.
Bien entendu, vous n'oublierez pas de créer les éventuelles relations !

3D. Formulaires

3D1. Introduction

C'est assez original… Quoi ? Et bien, nous n'avons pas besoin de formulaire pour entrer les données dans *Étudiant* puisque ces dernières viennent d'un import de fichier Excel. Enfin… certains champs doivent être renseignés (ceux identifiant les informations relatives au stage) ; nous le ferons plus tard.
Le seul formulaire est donc celui basé sur *Contact*. Nous allons successivement réaliser deux formulaires : un tout simple et un plus efficace.

3D2. Premier formulaire

C'est un entraînement… il doit permettre de saisir un enregistrement de *Contact* :

- je dois bien entendu pouvoir rentrer toutes les informations ;
- la saisie doit être conviviale (donc utiliser une zone de liste pour saisir l'étudiant).

Ce n'est guère plus qu'un formulaire généré par l'assistant et basé sur *Contact*. L'intérêt de ce formulaire est en fait de se rendre compte qu'il n'est pas efficace car toutes les informa-

tions ne sont pas accessibles : lorsque je saisis un contact pour un étudiant, je n'ai pas l'historique des précédentes conversations.
Faites donc ce formulaire en soignant l'ergonomie (dont la zone de liste).

3D3. Second formulaire

Celui-ci sera plus efficace. Il doit ressembler à l'état, à savoir donner la liste des différents contacts ayant déjà eu lieu. L'intérêt ? Quand un étudiant m'appelle, je le sélectionne et hop ! j'obtiens la liste de tous les contacts que nous avons déjà eus. Je peux donc me remémorer rapidement à qui j'ai affaire ! Pour faire cela, nous allons faire un couple formulaire/sous-formulaire :

- le formulaire principal présente l'étudiant et son entreprise ;
- le sous-formulaire liste les contacts avec cet étudiant ou son entreprise.

Dans un premier temps, vous vous contenterez de faire glisser les champs d'*Étudiant* dans le formulaire principal. Le problème, c'est que pour trouver un étudiant, il faut passer par les boutons de déplacement. Je vous propose donc dans un second temps d'ajouter une liste déroulante que vous paramétrerez comme *IndicatifE* de *Contact*. Ce contrôle sera appelé *RechercheNom* et sera **indépendant**. Il faudra donc le programmer pour le synchroniser vis-à-vis de l'enregistrement courant du formulaire :

- lorsque l'on ouvre le formulaire, il faut que la zone de liste affiche l'enregistrement correct. Pour cela, vous placerez le code suivant dans un événement précis du formulaire que je vous laisse chercher :

```
RechercheNom.Value = Me.RecordsetClone.IndicatifE
```

- lorsque l'on modifie la liste, l'enregistrement courant du formulaire doit être mis à jour. Je vous laisse le soin de trouver l'événement correspondant. Voici le code qu'il faut y mettre (pour comprendre ce qu'il signifie, étudiez l'aide ou… attendez l'année prochaine !) :

```
Me.RecordsetClone.FindFirst "IndicatifE = " & "'" & RechercheNom.Value & "'"
Me.Bookmark = Me.RecordsetClone.Bookmark
```

⚠ **Bien appeler votre zone de liste comme moi, sinon le code ne fonctionnera pas ! Ne cherchez pas à afficher la classe (1re ou 2nde année de BTS) : nous le ferons en 3F1.**

3E. État

Et bien, vous devez générer un état correspondant à l'annexe, sachant que l'état va donner l'intégralité des fiches de compte-rendu (une par étudiant). Notez bien que l'application est propre à un tuteur.

Une petite aide : vous devrez faire un état présentant l'étudiant et contenant deux sous-états (un pour les contacts avec l'étudiant, un pour les contacts avec l'entreprise).

3F. Pour fignoler…

3F1. Première optimisation : le formulaire

Le second formulaire est parfait… à un bémol près : je ne connais pas la classe de l'étudiant qui m'appelle. Or, mon discours ne sera pas le même selon qu'il est en première ou en seconde année. Il faut remédier à cela.

Il y a deux façons :

- on se contente d'afficher l'indicatif… l'utilisateur identifiera alors l'année en observant les trois premiers chiffres (782 pour la première année, 781 pour la seconde). Cette solution n'est évidemment pas acceptable ;
- on utilise un contrôle indiquant la classe.

Je vous demande d'appliquer cette seconde technique. Quelques indications :

- utilisez un contrôle adapté ;
- attention à une propriété très précise de ce contrôle qu'il faut absolument définir ;
- ce contrôle sera indépendant : sa valeur sera déterminée par programmation ;
- cette valeur sera définie pour chaque étudiant s'affichant. Il faut donc trouver un événement **du formulaire** se déclenchant lorsque l'on change d'enregistrement dans la source de données. Je vous laisse chercher quel est l'événement idoine… une petite aide : c'est un des premiers de l'onglet *Événement* ;
- je vous laisse le soin de trouver le code à écrire pour l'événement ci-dessus. Vous aurez besoin de la fonction *left* (voir l'aide) pour récupérer les trois premiers caractères de l'indicatif. Selon la valeur renvoyée par *left*, notre nouveau contrôle devra indiquer que l'étudiant est en première ou seconde année.

Pour faire le lien avec le support de cours, ce que je vous demande ici est assez simple mais est très rentable du point de vue de l'ergonomie.

3F2. Seconde optimisation : l'état

Pour économiser mon encre d'imprimante, les arbres de la planète et les frais de port, je ne veux pas imprimer les fiches de compte-rendu des étudiants pour lesquels il n'y a eu aucun contact. En effet, le suivi étant nul, la rémunération sera nulle.

Comment n'imprimer (ne générer) que les fiches des étudiants pour lesquels il y a eu au moins un contact ? C'est très simple, cherchez !

Annexe 1 : exemple de feuille Excel contenant la liste des étudiants

	A	B	C	D	E	F	G	H
1	**Indicatif**	**Nom**	**Prénom**	**Adresse**	**Code**	**Ville**	**Tél**	**Date de naissance**
2	782-81-0201-4	AL BYDEOI	Yannis	2 rue Vigne de Renard	01570	VILLEURBANNE	0130305315	10/05/1975
3	782-81-0202-5	ALBERTI	Yannick	8 rue Gauthier d' Epinal	01700	VILLENEUVE SUR	0134454084	23/02/1978
4	782-81-0203-6	AMOUROUX	Yann	21 rue de Lorraine	02220	VILLENEUVE SUR	0139568184	10/01/1977
5	782-81-0204-0	ASSILA	William	7 place d' Irlande	05000	VILLENEUVE LES	0141061053	16/04/1978
6	782-81-0205-1	AVRON	Vincent	81 bie av du Gal Leclerc	05190	VILLENEUVE LES	0145069855	18/06/1982
7	782-81-0206-2	AZRAK	Vincent	24 Impasse Dumont Durville	05380	VILLENEUVE D A	0145535815	25/03/1970
8	782-81-0208-4	BEHIER	Véronique	151 rue Gabriel Péri	06110	VILLEFONTAINE	0143050557	28/10/1975
9	782-81-0209-5	BEHRENS	Véronique	21 rue du Brocard	07410	VAUDONCOURT	0143035314	04/06/1980
10	782-81-0210-6	BERTHOLON	Valérie	53 rue de Morlaix	09420	VALENCIENNES	0143841130	10/02/1976
11	782-81-0211-0	BERTRAND	Tiana	16 rue du Grand Mont	12330	TRINITE	0145943015	03/07/1969
12	782-81-0212-1	BISMUTH	Thomas	586 rue du Grand-Chemin	13009	TOURS	0147098060	29/03/1966
13	782-81-0213-2	BONVAL	Thomas	rue de Versaille	13090	TOULOUSE	0147159580	24/04/1973
14	782-81-0214-3	BOUSSAA	Thierry	52 rue Claire Fontaine	14000	TOULOUSE	0148590533	18/05/1977
15	782-81-0215-4	BRUNO	Thierry	rue de Louvière	16300	TOULON	0148367549	03/04/1980
16	782-81-0216-5	CATRIN	Thierry	43 rue Louis Braille	17133	STRASBOURG	0153616590	30/08/1982
17	782-81-0217-6	CAUMETTE	Thibaut	Bât Granit E2 Appt 106	21083	STE CLOTILDE	0155458959	31/05/1977
18	782-81-0218-0	CHENG	Sylvain	5 Square de Bruges	22000	ST ROCH	0156081845	06/06/1970

Annexe 2 : fiche de compte-rendu

MINISTERE DE L'EDUCATION NATIONALE

CNED

COMPTE-RENDU
TUTORAT PEDAGOGIQUE

TUTEUR :
NOM : FEURIER
PRENOM : Jean-Yves
CODE PROFA : Techel clem

ETUDIANT
NOM : le Techel
PRENOM : Nina
INDICATIF : 8-149-312-27

Contacts engagés avec l'étudiant stagiaire :

Date	Nature de la demande
13/12	Premiers contacts
17/01	pb AP

Contacts engagés auprès de l'entreprise d'accueil :

13/02 : Pb convention
27/03 : Bilan stage

A Téquille , le 18/06/01
Signature

CENTRE NATIONAL D'ENSEIGNEMENT A DISTANCE
INSTITUT DE POITIERS-FUTUROSCOPE - TELEPORT 4 - BOULEVARD LEONARD DE VINCI - BP 51000 - 86980 FUTUROSCOPE CHASSENEUIL CEDEX
TEL : 05 49 49 96 00 - FAX : 05 49 49 04 18

Devoir auto-corrigé

Fiches de suivi des étudiants

1. Import du classeur Excel

Bon, il suffit de le faire… Je vous rappelle que l'on importe les données dans une nouvelle table que l'on complétera ensuite. Voici les réglages pour les différents écrans :

1. Il faut utiliser la feuille de données proposée *Liste étudiants tuteur FÉVRIER* (les autres sont vides) du classeur *Devoir Access auto-corrigé*.
2. Cochez la case *Première ligne contient les en-têtes*.
3. L'écran suivant est important : vous cliquez successivement sur l'en-tête de chaque champ et vous définissez son nom, son indexation et son type. Bien entendu, il faut renommer les en-têtes pour coller au MLD (notamment rajouter un *E* final à chaque champ et changer *Date de naissance* en *DateNaissanceE*). Il faut aussi indexer sans doublon *IndicatifE* puisque c'est l'identifiant. Tous les champs seront de type *Texte*, sauf la date de naissance (de type *Date/Heure*). *[Si ces changements ne sont pas faits lors de l'import, il est toujours temps de le faire plus tard à partir d'Access.]*
4. Choisissez en clé primaire *IndicatifE*.
5. La table s'appellera *Etudiant*.
6. L'importation doit alors se faire sans erreur.

2. Les tables et les relations

2A. Table Étudiant

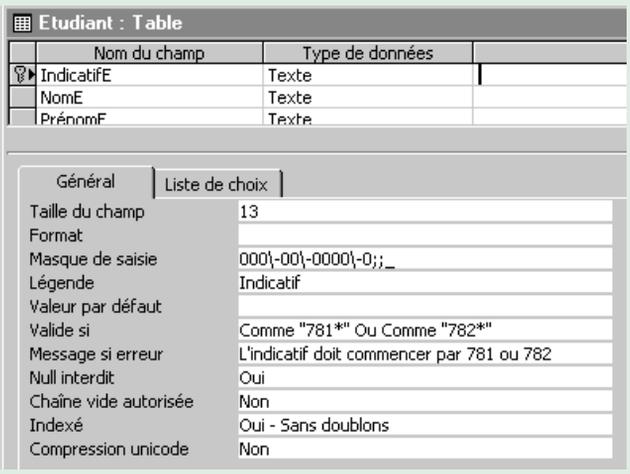
Il faut définir correctement les propriétés des champs créés par l'assistant d'importation et ajouter les champs définissant l'entreprise (*NomEnt*, *NomTuteurEnt* et *TélTuteurEnt*).

Nous allons aborder les principales difficultés des propriétés des champs.

Champ *IndicatifE*.

Je vous rappelle sa constitution : xxx-xx-xxxx-x. (trois chiffres, tiret, deux chiffres, tiret, quatre chiffres, tiret puis un chiffre).

Cela donnera les propriétés suivantes :



Quelques explications :

Masque de saisie :

Je l'ai fait avec l'assistant, en me basant sur le masque de l'ISBN, assez proche (des chiffres séparés par des tirets).

Valide si :

Je vous rappelle qu'un étudiant en BTS Info aura un identifiant commençant par 782 (1re année) ou 781 (2nde année). On mettra donc *Comme "781*" ou Comme "782*"*

Ce qui est important (et évalué) : la taille, le masque de saisie et la propriété *Valide si*. Bien entendu, *IndicatifE* doit être clé primaire.

Pour les champs *NomE*, *PrénomE*, *AdresseE*, *VilleE*, *NomEnt* et *NomTuteurEnt*, rien de particulier, si ce n'est la taille qu'il ne faut pas laisser à 255 ! (Je vous propose 20, sauf pour *AdresseE* où je mets 50.)
Le code postal (*CodeE*) aura une longueur de 5 et comme masque de saisie « 00000;;_ ».
Les deux téléphones (*TélE* et *TélTuteurEnt*) doivent avoir une longueur de 10 caractères et comme masque « 00\ 00\ 00\ 00\ 00;;_ ».
Enfin, les propriétés *IndicatifE*, *NomE*, *PrénomE*, *AdresseE*, *CodeE* et *VilleE* doivent absolument être renseignées (donc *Null interdit* doit être à *oui*). Les autres champs seront à *non* car l'étudiant ne donne pas toujours son téléphone (il peut être à l'étranger, en prison…) et les coordonnées de l'entreprise d'accueil ne sont pas connues en début d'année.

2B. Table Contact

Il y a moins de surprises. Il faut cependant faire attention aux points suivants :

- *Entreprise* sera un booléen. Je vous propose la valeur par défaut *non* car la majorité des contacts ont lieu avec l'étudiant ;
- *Nature* doit être un champ de type *Mémo* et surtout pas texte. En effet, vous y stockerez du texte résumant la teneur du contact. 255 caractères (longueur maximale d'un champ texte) pourrait donc être un peu limité ;
- *Date* aura pour valeur par défaut *Date()* (la date du jour car on stocke généralement les contacts en temps réel).

Tous les champs doivent être renseignés (donc *Null interdit* à *oui* pour tous).

N'oubliez pas que je voulais une liste modifiable pour saisir la valeur de *IndicatifE*. Il fallait affecter le type *Assistant liste de choix…* à ce champ puis se laisser guider. Mais, avant tout, comme je veux une saisie sympathique (la liste devra donc contenir par exemple *Février Jean-Yves*, soit le prénom concaténé au nom), je devrai baser ma zone de liste sur une requête :

```
select IndicatifE, NomE & " " & PrénomE as [Nom étudiant]
from Etudiant
order by 2;
```

N'hésitez pas à exécuter cette requête pour vérifier qu'elle correspond bien à vos besoins. Il est obligatoire de trier les étudiants pour que la saisie soit efficace.

J'appelle cette requête *Saisie de l'indicatif dans contact* et je peux alors lancer l'assistant :

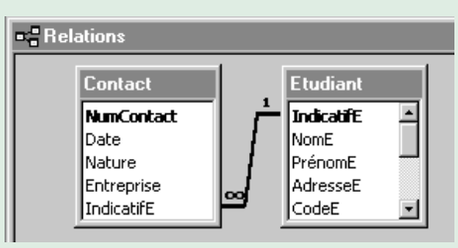
- 1er écran : choix 1 (les valeurs viennent d'une requête).
- 2e écran : choisissez la requête que nous venons d'écrire.
- 3e écran : on prend les deux champs (d'abord l'indicatif puis le nom).
- 4e écran : taille nulle à la colonne *IdentifiantE*, augmentez ensuite suffisamment la colonne contenant les noms.
- 5e écran : c'est *IndicatifE* que l'on veut stocker.
- 6e écran : je vous propose *Étudiant* comme étiquette.

L'intérêt de cela ? Et bien, dans le formulaire, l'essentiel du travail sera fait.

Évaluation : tout cela doit être fait, notamment ne pas oublier le tri dans la requête.

2C. Les relations

Il faut évidemment définir une relation avec intégrité référentielle entre les champs *IndicatifE* des deux tables :

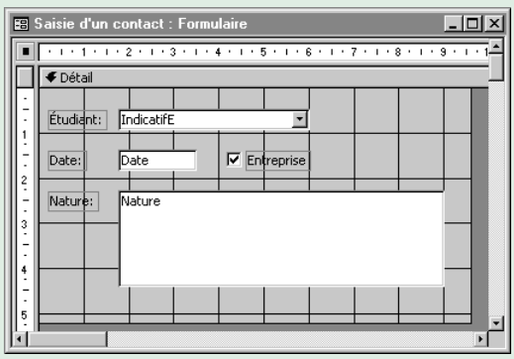


Évaluation : l'intégrité référentielle est indispensable.

3. Les formulaires

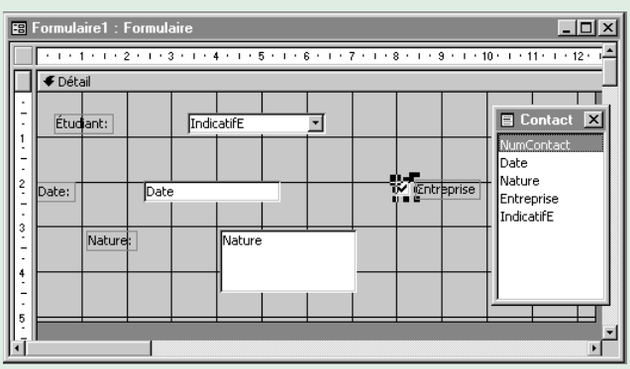
3A. Le plus simple

Comme les champs ont été définis correctement, il suffit de faire un formulaire vierge (*Mode Création*) basé sur *Contact* et d'y glisser les champs. Les contrôles adéquats seront utilisés. Ensuite seulement, on s'occupe de l'ergonomie.
D'abord, je glisse les champs (je ne prends pas *NumContact* car c'est un champ technique qui n'a aucun intérêt ici).



Vérifiez que les valeurs par défaut des champs *Date* et *Entreprise* ont bien été prises en compte ; sinon, définissez-les.

On peut ensuite placer les champs correctement ; voici le formulaire en mode *Création* puis *Formulaire*.



Je n'ai pas fait de zèle : pas d'image de petit teckel ni de boutons de déplacement. Je me contenterai donc de la barre d'état pour me déplacer dans les contacts et surtout du bouton d'ajout à chaque coup de téléphone.
Vous remarquerez que l'ajout de la liste modifiable ne m'a pas posé de problème : comme elle était utilisée dès la définition de la table, Access a de lui-même repris ce contrôle dans le formulaire.
Le problème est que ce formulaire ne me donne pas l'historique de mes contacts avec Nathalie Guilhamat (ou les autres étudiants). Pour trouver les contacts précédents que nous avons éventuellement eus, il faut utiliser les boutons de déplacement, ce qui n'est pas très convivial.
La seconde solution avec le sous-formulaire sera plus agréable !

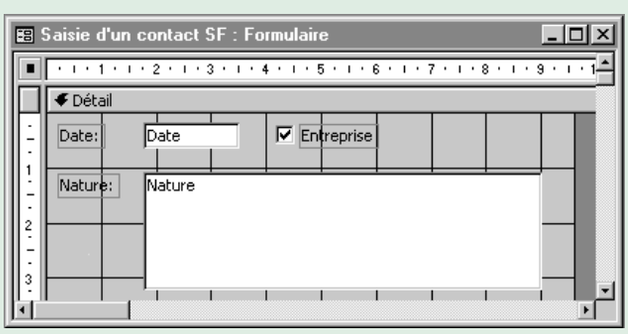
3B. Le plus efficace

3B1. Sous-formulaire

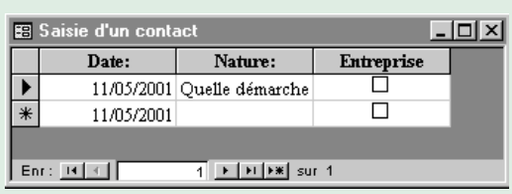
D'abord, le futur sous-formulaire. Il est basé sur *Contact* comme le précédent. La seule différence ? Il est en mode *Feuille de données* et l'indicatif de l'étudiant n'apparaît plus puisque cette information sera indiquée dans le formulaire principal. Bien entendu, l'indicatif doit néanmoins faire partie de la source de données alimentant le sous-formulaire puisqu'il permettra de faire la synchronisation avec le formulaire principal.
Nous pourrions créer le sous-formulaire de toutes pièces comme celui fait en 3A. Je vous propose une solution plus astucieuse : faites un copier/coller du formulaire 3A. La démarche est classique. Depuis la fenêtre *Base de données* (*F11*), sélectionnez les objets *Formulaires*, cliquez sur celui fait en 3A, puis copier/coller et donnez un nom explicite, par exemple *Saisie d'un contact SF*.
Ouvrez ce nouveau formulaire en mode *Création*. Basculez sa propriété *Affichage par défaut* (onglet *Format*) à *Feuille de données*. Supprimez ensuite le contrôle *Étudiant* inutile (vous pouvez vérifier que la source est bien *Contact*, donc l'indicatif sera accessible).

On obtient :

mode *Création*

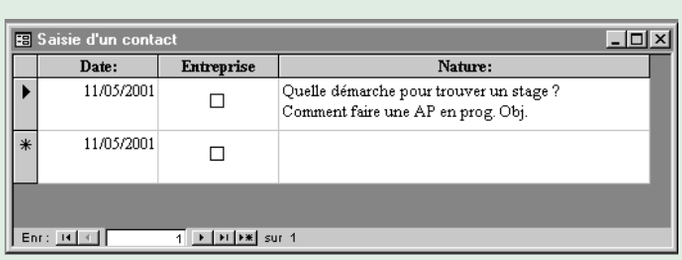


mode *Formulaire*



Seul problème : je préférerais permuter les colonnes *Entreprise* et *Nature* et allonger cette dernière. C'est très facile, c'est comme sous Excel pour redimensionner. Pour déplacer, cliquez dans l'en-tête de la colonne à partir du mode *Formulaire* puis re-cliquez en laissant appuyé… vous déplacez la colonne.

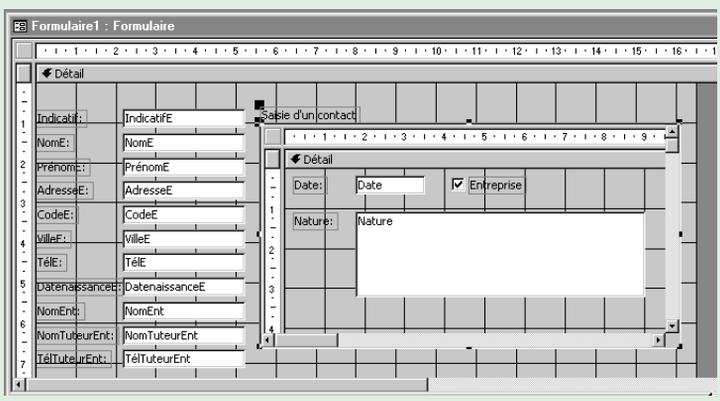
Enfin, j'augmente la hauteur des lignes (comme sous Excel) pour que le texte de *Nature* soit plus visible. On obtient :



3B2. Le formulaire principal

Il doit afficher toutes les informations propres à l'étudiant (ses coordonnées et celles de l'entreprise). Il est donc tout naturellement basé sur la table *Étudiant*. On lui ajoute ensuite le sous-formulaire fait précédemment en le glissant depuis la fenêtre *Base de données*.

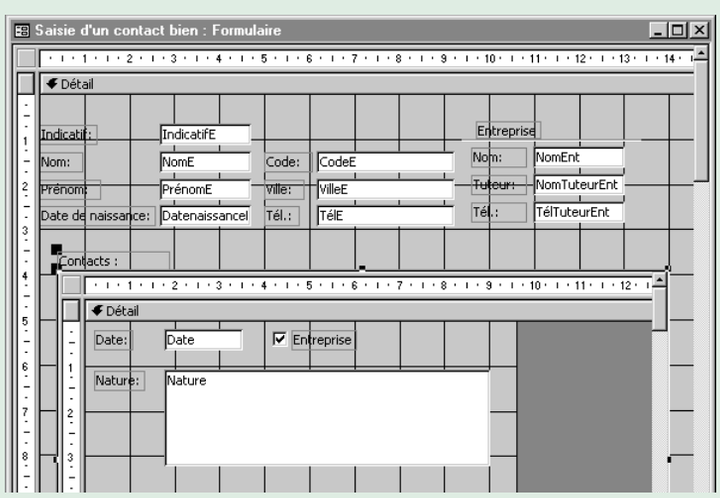
Première étape, je glisse tous les champs et le sous-formulaire :



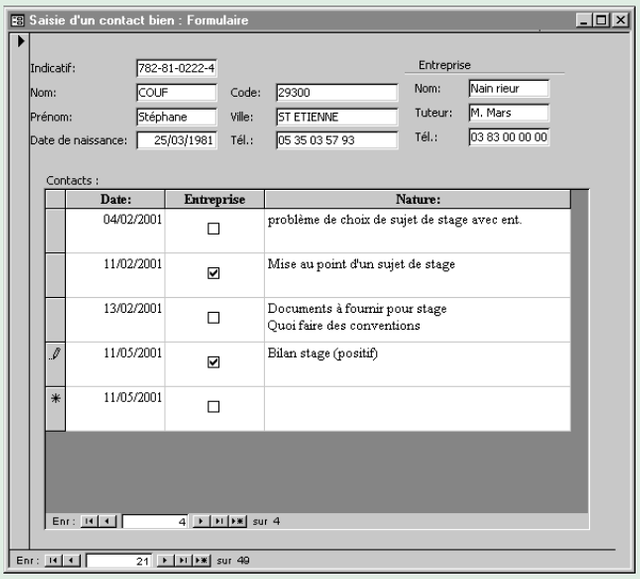
Deuxième étape, je vérifie que la jointure entre les deux formulaires est correcte (champs *IndicatifE* des deux sources). C'est une propriété du contrôle sous-formulaire :

Troisième étape, je réorganise un peu les contrôles (j'enlève *AdresseE* qui ne me sert à rien mais je garde le département et la ville car ce peut être utile lors de conseils sur la recherche de stage).

En mode *Création* :



En mode *Formulaire* :



Et bien, c'est parfait… Sauf que lorsqu'un étudiant téléphone, je dois utiliser les boutons de déplacement pour trouver sa fiche. J'aimerais retrouver la fonctionnalité de liste déroulante utilisée dans le paragraphe 3A. Pourquoi ne l'avons-nous plus ? Car dans le formulaire principal, l'indicatif affiché vient de la table *Étudiant*. Pour ce champ clé primaire, nous n'avons évidemment défini aucune zone de liste.

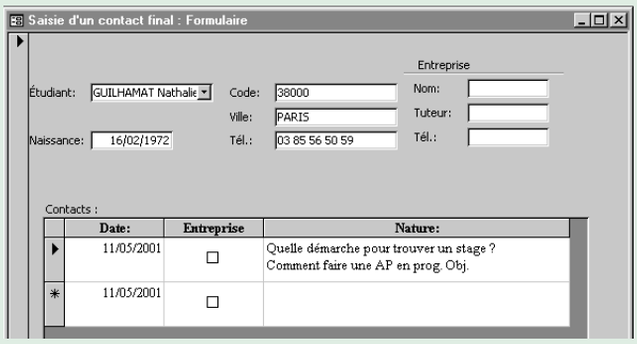
Nous allons donc ajouter un contrôle *Zone de liste déroulante* en le paramétrant comme dans 2B. Ce contrôle sera indépendant et les codes fournis dans le sujet seront associés aux événements *Sur chargement* du formulaire et *Après MAJ* pour la zone de liste comme l'illustre la copie suivante.

```
Private Sub Form_Load()
   RechercheNom.Value = Me.RecordsetClone.IndicatifE
End Sub

Private Sub RechercheNom_AfterUpdate()
   Me.RecordsetClone.FindFirst "IndicatifE = " & "'" & RechercheNom.Value & "'"
   Me.Bookmark = Me.RecordsetClone.Bookmark
End Sub
```

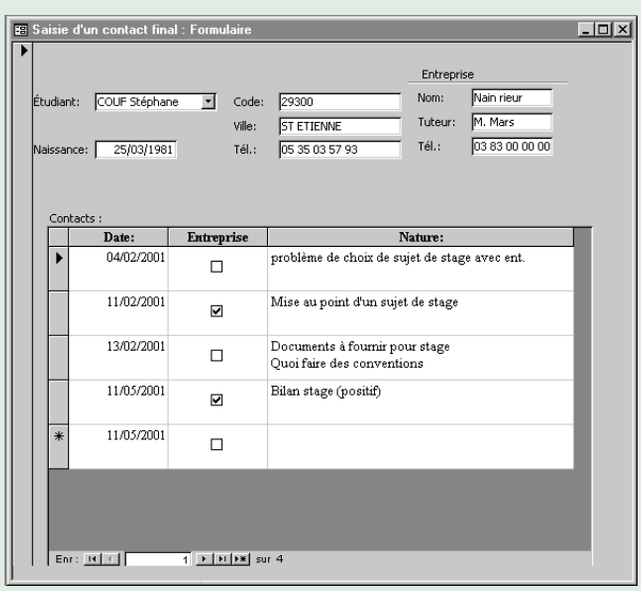
On peut désormais supprimer les contrôles affichant le nom et le prénom. Il fallait aussi penser à supprimer les boutons de déplacement du formulaire principal car la navigation ne doit être faite que par la zone de liste (sinon, on désynchroniserait la liste avec le formulaire et il faudrait utiliser un nouvel événement du formulaire pour y remédier. Pardon ? Quel événement ? Vous le chercherez dans le paragraphe 4E1).

On obtient alors :



En sélectionnant un nouvel étudiant :

J'obtiens directement sa fiche :



4. L'état

Il est plus simple que le formulaire car nous n'aurons pas à nous casser la tête avec les contrôles. La difficulté est néanmoins présente : il faut savoir dans quelle zone mettre chaque information. Je vous rappelle les hypothèses de travail avec leurs conséquences.

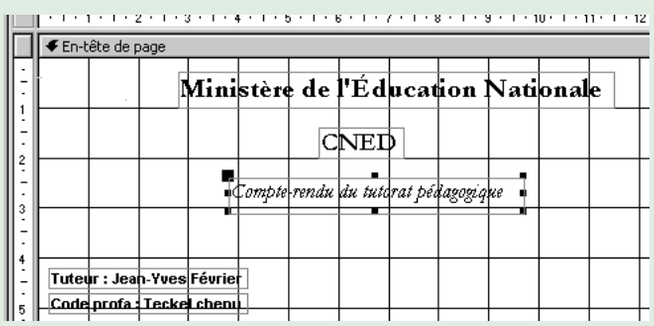
Contraintes	Conséquences
Une feuille imprimée par étudiant	Toutes les informations liées à l'étudiant seront dans la zone *Détail*
Mon nom et mon *code préfa* doivent être sur chaque feuille.	Je les mets dans la zone *En-tête de page*
Le nom et l'indicatif de l'étudiant doivent être imprimés	RAS
Les contacts avec l'étudiant doivent être imprimés	J'utiliserai un sous-état dans la zone *Détail* chargé de n'afficher que les contacts avec l'étudiant
Les contacts avec l'entreprise de l'étudiant seront imprimés	Idem, sous-état dans la zone *Détail* mais n'affichant que les contacts avec l'entreprise de l'étudiant.
Chaque feuille doit comprendre le lieu et la date	Le lieu, c'est chez moi (*Téquelle*), la date est celle du jour et je mettrai tout cela dans le pied de page

4A. Préparation

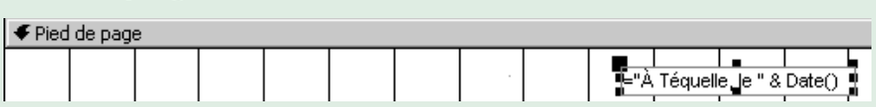
On génère un état en *Mode Création* basé sur *Étudiant*. Ce sera notre état principal.

4B. En-tête

Toutes les informations dans l'en-tête sont des constantes, notamment mon nom et mon *code profa* puisque cette application m'est propre : je suis une constante de l'application (comme tout autre tuteur l'utilisant).



4C. Le pied de page



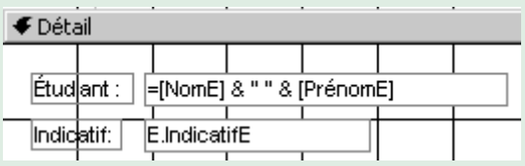
Sans commentaire…

4D. Détail

J'y mets trois choses : les coordonnées de l'étudiant puis les deux sous-états.

4D1. Coordonnées

Aucun problème, il suffit de faire glisser les champs utiles et de mettre à jour les libellés. Je mets une petite formule pour afficher le nom et le prénom ensemble.



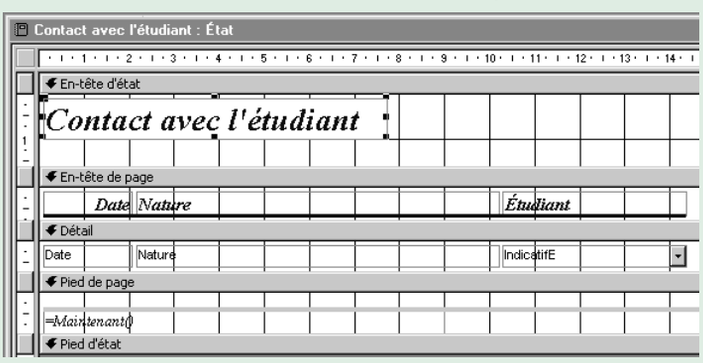
4D2. Premier sous-état : contacts avec l'étudiant

Je veux la liste des contacts, donc voici la requête sur lequel le sous-état sera basé :

```
SELECT Date, Nature, IndicatifE
FROM Contact
WHERE Not (Entreprise)❶
ORDER BY 1;
```

Bien entendu, on trie sur les dates ! Notez que je dois renvoyer *IndicatifE* car je joindrai état et sous-états sur ce champ (exactement comme avec le sous-formulaire vu en 3B).

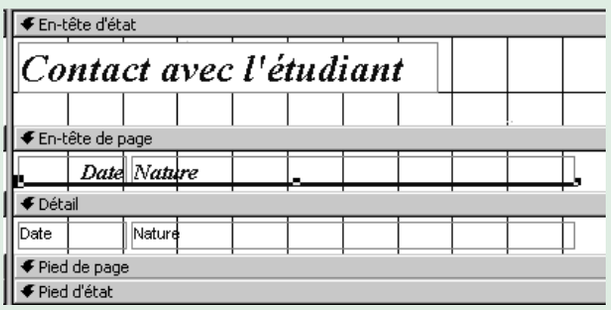
On crée ensuite le sous-état *Contact avec l'étudiant*. C'est un état comme un autre, très vite fait : on fait un état instantané en tableau basé sur la requête ci-dessus.



On enlève le champ *IndicatifE*, qui ne servira qu'à faire le lien entre état et sous-état (il ne sera donc plus affiché, mais fait toujours partie de la source de données donc reste accessible).

❶ WHERE Not(Entreprise) *peut s'écrire* WHERE Entreprise = false. *C'est moins élégant mais peut-être plus clair pour vous.*

On enlève également le pied de page sans intérêt pour un sous-état. On obtient :



4D3. Second sous-état : contacts avec l'entreprise

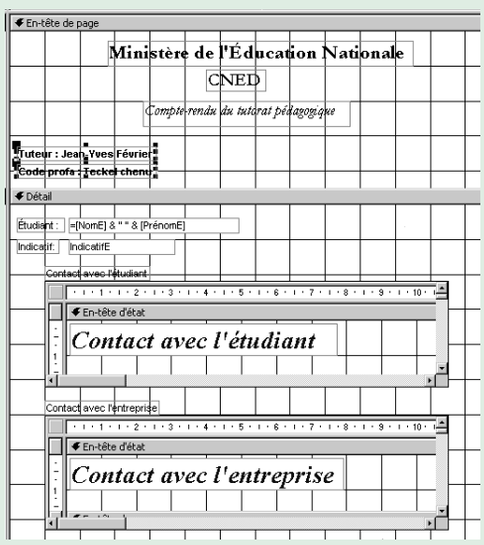
Je veux la liste des contacts, donc voici la requête sur laquelle le sous-état sera basé :

```
SELECT Date, Nature, IndicatifE
FROM Contact
WHERE Entreprise❶
ORDER BY 1;
```

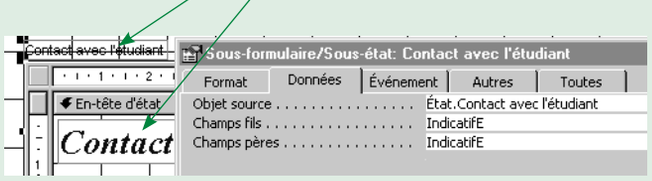
La modification est minime ! On fait ensuite comme pour le premier sous-état.

4D4. Intégrons les sous-états

C'est très simple : la démarche est la même que pour les sous-formulaires. On fait glisser les futurs sous-états depuis la fenêtre *Base de données* vers l'état principal en mode *Création*. On obtient :



Il faudra enlever les **étiquettes** des deux sous-états puisqu'ils possèdent leur propre **titre**.



On vérifie que les deux sous-états sont correctement joints avec l'état principal (voir copie d'écran ci-dessus).

❶ *De même,* WHERE Entreprise *peut s'écrire* WHERE Entreprise = true.

Au final, cela fonctionne très bien. Voici une page représentative :

Ministère de l'éducation nationale

CNED

Compte rendu du tutorat pédagogique

Tuteur : Jean-yves Février
Code prof. : Teckel chenu
Étudiant : Couf Stéphane
Indicatif : 782 5702650

Contacts avec l'étudiant

04/02/2001	Problème de choix de sujet de stage
11/02/2001	Mise au point d'un sujet de stage
13/02/2001	Documents à fournir pour le stage
	Quoi faire des conventions

Contacts avec l'entreprise

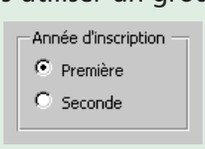
11/05/2001	Bilan de stage (positif)

À Téquelle, le 11/05/2001

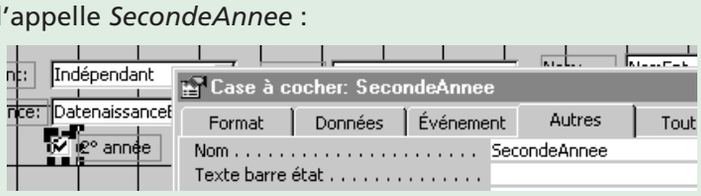
4E. Fignolons…

4E1. Connaître l'année de l'étudiant

Quel contrôle utiliser ? Nous pourrions utiliser un groupe d'options :



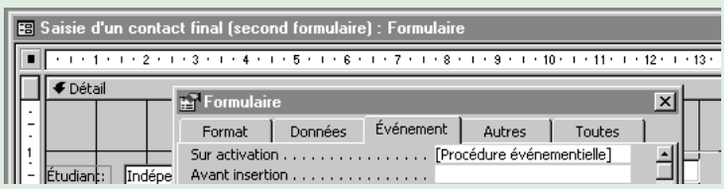
Mais bon, ce contrôle prend de la place sur le formulaire et son ergonomie est mitigée : en effet, le choix est binaire (première ou seconde année). Je vous propose donc d'utiliser une case à cocher. Je l'appelle *SecondeAnnee* :



La propriété importante à définir, c'est *Verrouiller* qu'il faut mettre à *Oui* (par défaut, c'est à

Non). En effet, comme pour tout contrôle indépendant dont la valeur est définie par calcul, l'utilisateur ne doit pas pouvoir la changer.
L'événement du formulaire qui nous intéresse, c'est *Sur activation* (le premier de la liste). Il se déclenche à chaque fois que l'enregistrement courant de la source de données (donc l'enregistrement affiché) change. C'est bien l'événement nécessaire puisqu'à chaque nouvel étudiant, il faut déterminer s'il est ou non en seconde année.

On associe donc du code à cet événement comme l'illustre la copie d'écran suivante.

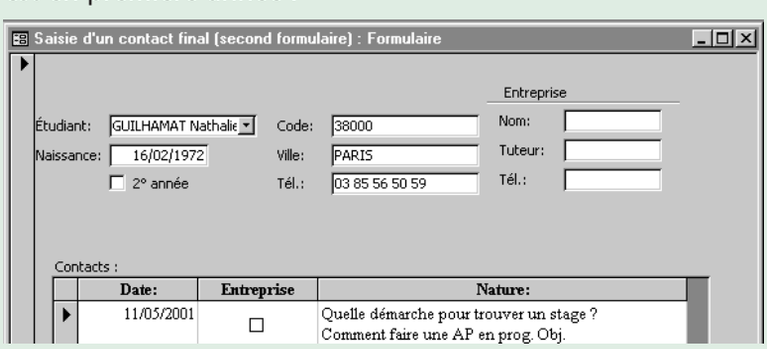


Le code est sans surprise :

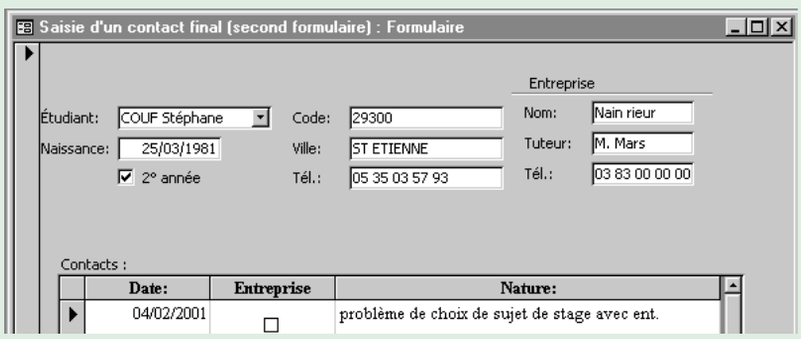
```
Private Sub Form_Current()
  If Left(IndicatifE, 3) = "781" Then
      SecondeAnnee.Value = True
    Else
      SecondeAnnee.Value = False
  End If
End Sub
```

Voilà ! Vérifions que cela fonctionne :

M^lle^ Guilhamat est en première année :



M. Couf est en seconde année :



4E2. Que les étudiants ayant réalisé un contact

C'est très simple : pour n'avoir dans l'état que les étudiants avec qui j'ai eu au moins un contact, il me suffit de le demander ! Comment ? Et bien, en basant mon état principal non plus sur la liste des étudiants (table *Étudiant*) mais sur la liste des étudiants ayant réalisé au moins un contact. Mon état sera donc basé sur la requête suivante :

```
select *
from Etudiant E
where exists (select *
                from Contact C
                where E.IndicatifE = C.IndicatifE) ;
```

C'est tout !

5. Voilà, c'est fini !

Nous avons fait une application restreinte mais complète et sur un sujet réel. J'espère que cela vous a permis de bien cerner Access.

Vous pouvez, maintenant, faire et envoyer à la correction le devoir 4 (voir fascicule « Devoirs » réf. 3932 DG)