Base de données bureautique avec Access

Cyril Gruau

15 mai 2006

Résumé

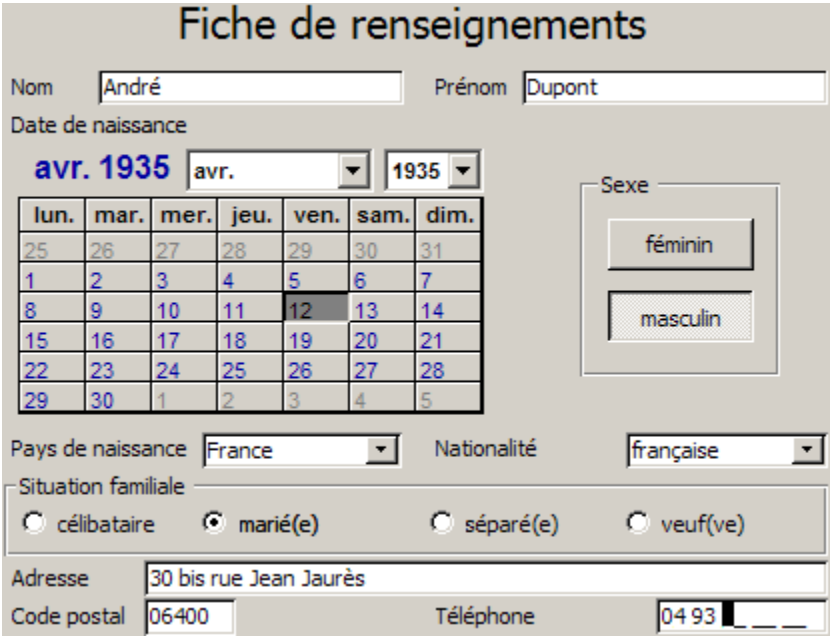
Parmi les logiciels de bureautique, Word permet d'éditer des textes, Excel d'élaborer des tableaux et des graphiques, PowerPoint de structurer des présentations et Access sert à créer des fiches de renseignements, ainsi qu'à éditer des états. Ces fonctionnalités exigent, de nous, un approfondissement en matière d'organisation du stockage de nos données et de rigueur en ergonomie dans nos formulaires.

Fiche de renseignements

Nom : ..... Prénom : .....
Date de naissance : . . / . . / . . . . Sexe : ☐ F ☐ M
Pays de naissance : ..... Nationalité : .....
Situation familiale[1] : célibataire - marié(e) - séparé(e) - veuf(ve)
Adresse : .....
Code postal : ..... Téléphone : .....

[1] entourer la mention utile

(a) formulaire papier



(b) équivalent bureautique

Fig. 1 – *Remplacement d'un formulaire classique par un formulaire Access*

Table des matières

Introduction **5**

1 Tables **6**
1.1 Types de données . . . 7
1.2 Affichage et masque de saisie . . . 9
1.3 Clé primaire . . . 11
1.4 Contraintes . . . 12

2 Clés étrangères **14**
2.1 Normalisation . . . 15
2.2 Liaison un-à-plusieurs . . . 16
2.3 Liaison plusieurs-à-plusieurs . . . 19
2.4 Deuxième forme normale . . . 20
2.5 Gestion des dates et du caractère historique . . . 22
2.6 Liaison un-à-un . . . 23
2.7 Héritage . . . 25
2.8 Auto-liaison . . . 26
2.9 Liaison composite . . . 27
2.10 Sous-numérotation . . . 28
2.11 Conclusion . . . 29

3 Requêtes de sélection **30**
3.1 Sélection simple . . . 31
3.2 Propriétés de requête . . . 32
3.3 Sélection sur plusieurs tables . . . 34
3.4 Vues . . . 35
3.4.1 Ré-utilisation d'une requête dans une autre requête . . . 35
3.4.2 Jointure . . . 36
3.4.3 Conclusion . . . 37
3.5 Opérations . . . 37
3.6 Critères . . . 38
3.7 Groupements . . . 40
3.7.1 Critères sur les lignes et méthodologie . . . 41
3.7.2 Informations complémentaires . . . 42
3.7.3 Regroupements sur plusieurs colonnes . . . 42
3.7.4 Vues préalables . . . 43
3.7.5 Conclusion . . . 43
3.8 Auto-jointure . . . 44
3.9 Jointures externes . . . 45
3.9.1 Remplacement des valeurs vides . . . 46
3.9.2 Jointure externe plusieurs-à-plusieurs . . . 47
3.9.3 Auto-jointures externes . . . 48
3.9.4 Élimination des cycles . . . 49
3.10 Conclusion . . . 50

4 Requêtes d'action **51**
4.1 Insertion (ajout) . . . 52
4.2 Suppression . . . 56
4.3 Mise à jour . . . 57
4.4 Création de table . . . 59

5 Formulaires **61**
5.1 Propriétés d'un formulaire . . . 62
5.2 Principaux éléments constitutifs . . . 64
5.3 Contrôles . . . 66
5.3.1 Correction du formulaire généré par l'assistant . . . 66
5.3.2 Groupe d'options . . . 68
5.3.3 Liste déroulante . . . 70
5.3.4 Toupie par zone de liste . . . 72
5.3.5 Sélection multiple et listes duales . . . 73
5.3.6 Compléments sur les listes . . . 74
5.4 Sous-formulaire . . . 75
5.5 Requête source . . . 77
5.6 Ergonomie . . . 78
5.7 Recherche, tri et filtre . . . 80
5.8 Conclusion . . . 81

6 États **82**
6.1 Création . . . 83
6.1.1 Composition . . . 84
6.1.2 Requête source . . . 85
6.1.3 Utilisation de l'assistant . . . 86
6.1.4 Correction de l'état obtenu . . . 87
6.1.5 Gestion des groupes et des tris . . . 90
6.2 Annuaire . . . 90
6.3 Mise en forme conditionnelle . . . 94
6.4 Sous-états . . . 96
6.4.1 Création de l'état principal . . . 97
6.4.2 Création des deux sous-états . . . 98

7 Visual Basic for Applications **100**
7.1 Description du langage de programmation . . . 101
7.1.1 Types de données . . . 101
7.1.2 Déclaration et affectation . . . 102
7.1.3 Opérateurs et affichage . . . 102
7.1.4 Procédures et fonctions . . . 103
7.1.5 Structures de décision . . . 105
7.1.6 Structures de répétition . . . 107
7.2 Objets et événements . . . 109
7.2.1 Contrôles . . . 109
7.2.2 Formulaires et états . . . 111
7.2.3 Collections . . . 113
7.2.4 L'objet `DoCmd` . . . 114
7.3 Erreurs et débogage . . . 118
7.3.1 Déboguer . . . 118
7.3.2 Piéger les erreurs . . . 121

8 VBA et SQL **122**
8.1 Requêtes paramétrées . . . 122
8.2 Domaines . . . 124
8.2.1 Zone de texte calculée . . . 124
8.2.2 Critère fondé sur une liste à sélection multiple . . . 125
8.2.3 Sous-numérotation . . . 126
8.3 Filtres . . . 127
8.4 Gérer une liste à sélection multiple . . . 129

8.4.1 Dans un formulaire de saisie 130
8.4.2 Dans un formulaire de modification 132
8.5 Gérer deux listes duales 134
8.6 Ajouter une valeur manquante dans une liste déroulante 138

9 Application autonome **139**
9.1 Menu général 139
9.2 Menus personnalisés 142
9.3 Identification 145
9.4 Fractionnement et compilation 149

Conclusion **150**

Remerciements **150**

Références **151**

Table des figures **154**

Index **155**

Introduction

Le logiciel Access n'est pas un SGBD[1] à proprement parler, mais un outil de bureautique qui permet de créer des formulaires, comme par exemple, des fiches de renseignements (figure 1).

Il se trouve que l'exploitation informatique de tels formulaires nécessite, au préalable, la présence :

- de tables, rigoureusement liées entre elles par des clés étrangères, pour stocker les données qui sont saisies au travers des formulaires ;
- et des requêtes, qui permettent de retrouver les données stockées dans les tables, puis d'en insérer de nouvelles, d'en supprimer ou d'en mettre à jour une partie.

Afin d'être un logiciel autonome, Access fournit donc, non seulement les outils pour mettre en place des formulaires (section 5), mais aussi les services rudimentaires d'un SGBDR[2] (sections 1 à 4).

Ainsi, outre les notions de développement objet (langage Visual Basic for Applications, sections 7 à 9), qu'il faut connaître pour construire les formulaires (surtout lorsque leur fonctionnement devient complexe), de nombreuses notions relationnelles, comme la conception d'un schéma relationnel ou l'élaboration de requêtes SQL (pour Structured Query Langage), sont nécessaires à la maîtrise de ce logiciel.

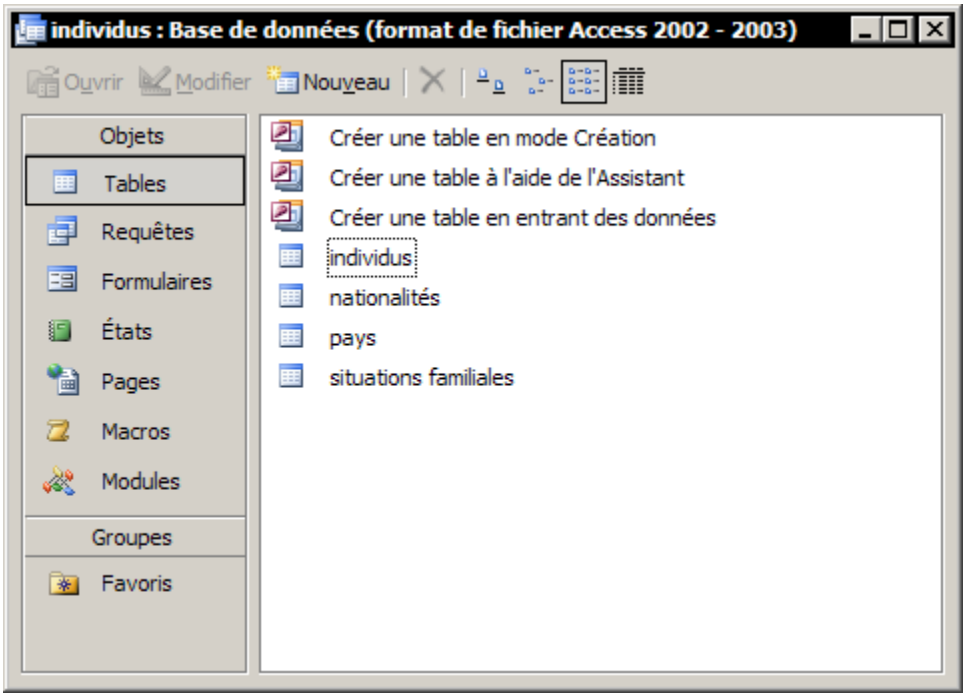


FIG. 2 – *Fenêtre d'accueil du logiciel*

À ces notions s'ajoutent celles du reporting, indispensables pour savoir utiliser les fonctionnalités d'édition d'états (section 6) qui, accompagnées des pages web, font d'Access un logiciel de développement d'applications complètes (figure 2), destinées à la fois à la production de données, à leur traitement automatique et à leur analyse.

1. Système de Gestion de Bases de Données
2. R pour Relationnelles

1 Tables

Les données saisies au travers des formulaires sont stockées dans des tables qui nécessitent, au préalable, un travail rigoureux de conception, afin de chasser toute redondance d'information et d'assurer l'intégrité des données.

Considérons le formulaire de la figure 1 page 1 : pour stocker les données qui sont saisies à travers ce formulaire, une table des individus est bien sûr nécessaire (figure 3(a)).

Nom du champ	Type de données
numéro individu	NuméroAuto
nom	Texte
prénom	Texte
homme	Oui/Non
date naissance	Date/Heure
numéro pays naissance	Numérique
numéro nationalité	Numérique
numéro situation familiale	Numérique
adresse	Texte
code postal	Texte
téléphone	Texte

(a) table des individus

Nom du champ	Type de données
numéro situation familiale	NuméroAuto
situation familiale	Texte

(b) table de codification

FIG. 3 – *Tables en mode création et définition des colonnes*

Mais il ne faut pas oublier d'ajouter une table de codification pour répertorier, une bonne fois pour toutes, les situations familiales (figure 3(b)). Les avantages de disposer d'une telle table de codification sont multiples :

- les choix possibles sont saisis une bonne fois pour toute, limitant ainsi les fautes de frappes et le travail de correction éventuel ;
- il est également possible d'ajouter sans difficulté une nouvelle alternative qui n'avait pas été prévue au départ (comme `pacsé(e)` qui n'était pas prévu sur la figure 1), en ajoutant une ligne dans la table de codification, puis en le répercutant dans le formulaire (manuellement pour les groupes d'options, malheureusement) ;
- les choix possibles sont numérotés automatiquement (grâce à la clé primaire de la table de codification) et seul ce numéro est stocké dans la table `individus` (plus précisément, dans la colonne `numéro situation familiale`, qui est dite clé étrangère), ce qui représente un gain de place substantiel par rapport au stockage de la chaîne de caractères elle-même.

Pour les pays et les nationalités, il convient également de prévoir des tables de codification, de sorte que le formulaire de la figure 1 requiert quatre tables au total (figures 2 et 4).

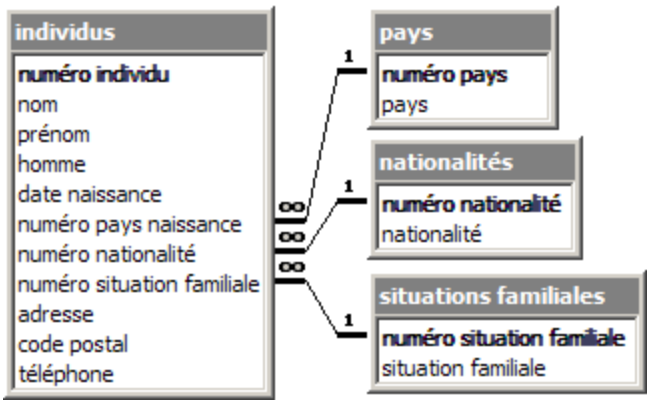


FIG. 4 – *Schéma relationnel pour un formulaire de renseignement*

La première étape dans le processus de création d'un fichier Access, consiste donc à décrire la structure de ses tables. Créer une nouvelle table consiste, pour chaque colonne de la table, à préciser son intitulé[3], son type de données et ses contraintes éventuelles, notamment la clé primaire.

1.1 Types de données

Attribuer à chaque colonne le type de données (tableau 1) qui lui convient, est une étape facile. Encore faut-il prendre le réflexe de choisir :

- pour les le type `NuméroAuto` ;
- par voie de conséquence, pour les le type `Numérique` ;
- et pour les numéros de téléphone et les codes postaux, qui sont des nombres pouvant commencer par 0, le type `Texte` (avec un masque de saisie approprié).

type	description	exemples d'utilisation
`Texte`	chaîne d'au plus 255 caractères (au delà, utiliser le type `Mémo`)	un nom, une adresse, une description (le texte intégral d'un article de journal)
`Numérique`	entier ou réel à virgule flottante	un nombre, une quantité, une valeur, une grandeur physique
`Monétaire`	réel à virgule fixe (4 décimales après la virgule)	un prix unitaire, un montant total, un prix au litre
`Date/Heure`	année/mois/jour et/ou heure:minute:secondes	une date de naissance, une heure de départ
`Oui/Non`	booléen	un adjectif (qui peut être vrai ou faux), une dichotomie (homme ou femme)
`NuméroAuto`	entier incrémenté automatiquement	une clé primaire classique (qui n'est pas clé étrangère)

TAB. 1 – *Principaux types de données Access*

3. dans ce document, nous choisissons volontairement des intitulés explicites (non codés), comportant éventuellement des accents et des espaces, ce que le logiciel gère parfaitement bien (profitons-en)

Pour les colonnes de type `Numérique`, le choix de la taille du champ est fondamental car c'est à cet endroit que la différenciation entre les entiers et les réels à virgule flottante se fait (figure 5).

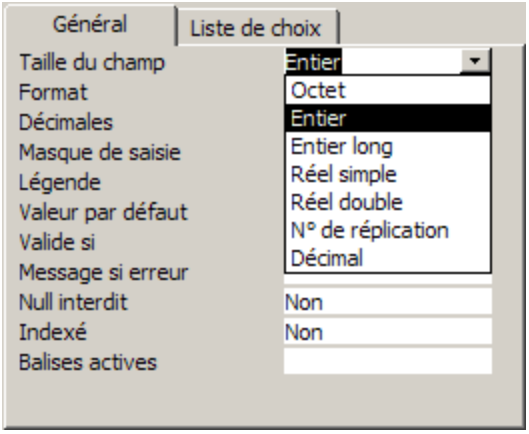


FIG. 5 – *Liste des sous-types à choisir pour le type* `Numérique`

C'est aussi l'occasion de décider de la taille (en octets ou en caractères, qui sont des unités équivalentes) occupée par la colonne (tableau 2).

sous-type	description	taille
`Octet`	entier entre 0 et 255	1 caractère
`Entier`	entier entre $-32\,768$ et $32\,767$	2 caractères
`Entier long`	entier entre $-2\,147\,483\,648$ et $2\,147\,483\,647$	4 caractères
`Réel simple`	réel à virgule flottante avec 7 décimales exactes entre $\pm 1.401298\,10^{-45}$ et $\pm 3.402823\,10^{38}$ (et 0)	4 caractères
`Réel double`	réel à virgule flottante avec 15 décimales exactes entre $\pm 4.94065645841247\,10^{-324}$ et $\pm 1.79769313486231\,10^{308}$ (et 0)	8 caractères

TAB. 2 – *Principaux sous-types du type* `Numérique`

Pour les chaînes de caractères, il est également important d'adapter la taille maximale au contenu, car la valeur par défaut (50 caractères) convient rarement. Elle ne suffit pas pour une description (espace insuffisant), tandis qu'elle est trop élevée pour un prénom (gaspillage d'espace mémoire).

1.2 Affichage et masque de saisie

Pour des grandeurs physiques, le type `Numérique` et le sous-type `Réel simple` peuvent être complétés de la description du format d'affichage qui permet notamment d'afficher les unités. Par exemple, pour une colonne `poids`, il est conseillé d'afficher automatiquement l'unité, ici le kilogramme, à l'aide du réglage de la figure 6.

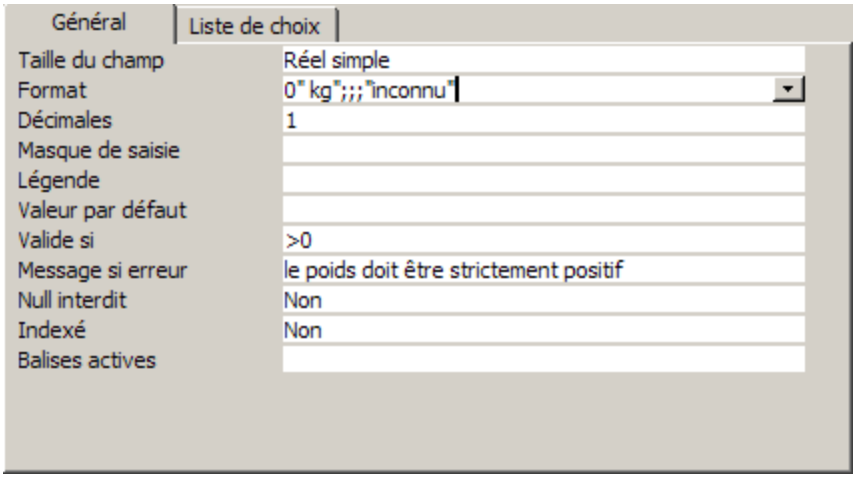


FIG. 6 – *Définition d'un format d'affichage*

Pour une valeur de type `Numérique`, nous pouvons définir quatre formats distincts, séparés par des points-virgules :

- le premier format est dédié aux valeurs positives (ici, nous avons ajouter l'unité `kg` précédée d'une espace) ;
- le deuxième format aux valeurs négatives (ici, le poids n'est pas censé être négatif) ;
- le troisième format à la valeur nulle (ici, le poids n'est pas censé être nul) ;
- le dernier format à la valeur vide (ici, nous en profitons pour remplacer les valeurs vides par la chaîne `inconnu`).

Sur cet exemple nous n'avons pas changé la couleur d'affichage, mais nous aurions pu. Cela permet notamment de mettre en rouge les valeurs négatives. Le lecteur peut consulter la référence [Isoz] pour découvrir cette fonctionnalité.

Ce format ne change rien à la façon dont sont stockées les valeurs. Mais il permet d'en personnaliser l'affichage, ce qui sera automatiquement répercuté dans les formulaires et les états. Les caractères spéciaux utilisables dans la définition d'un format d'affichage sont bien décrits dans l'aide du logiciel.

Notons que le nombre de décimale(s) a été volontairement positionné à 1 sur la figure 6 afin de limiter le nombre de chiffre(s) affiché(s) (et non pas stockés) après la virgule.

Le lecteur est invité à consulter la liste des formats prédéfinis par Access (dans la liste déroulante de la rangée `format`), non seulement pour le type de données `Numérique`, mais aussi pour le type `Date/Heure`. Ces formats prédéfinis suffisent généralement à satisfaire nos besoins.

Par ailleurs, afin de faciliter la saisie des données dans le bon format, il est conseillé d'utiliser les masques de saisie prédéfinis par Access. C'est très utile pour la saisie d'une date, d'une heure, d'un code postal français ou encore d'un numéro de téléphone. Le choix d'un masque se fait en affichant la fenêtre de la figure 7, à l'aide du bouton aux trois petit points ...

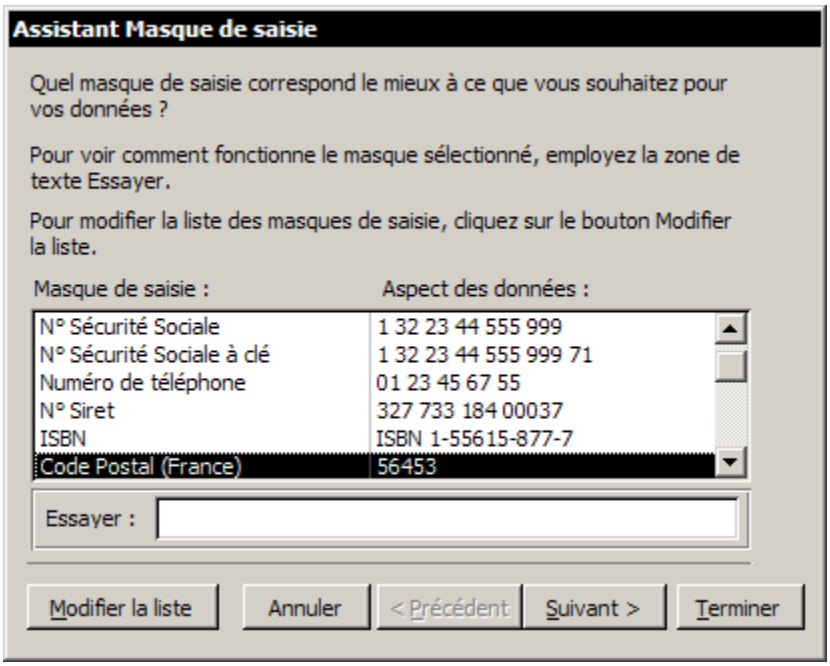


FIG. 7 – *Masques de saisie prédéfinis*

De tels masques de saisie peuvent être personnalisés manuellement, à l'aide de caractères spéciaux dont la description complète peut être trouvée dans l'aide d'Access.

1.3 Clé primaire

Dans une base de données pour une bibliothèque, il est nécessaire de prévoir une table pour les livres. Malheureusement, à la table des livres dont les types de données, les intitulés et les premières lignes sont décrits dans le tableau 3, il manque une colonne indispensable. Il s'agit de la clé primaire.

types	Texte	Texte	Date/Heure	Numérique	Oui/Non
intitulés	titre	auteur(s)	date d'acquisition	nombre de pages	emprun-table
	Ça	Stephen King	12/04/1999	234	Non
lignes	Code Da Vinci : l'enquête	M.-F. Etchegoin et F. Lenoir	17/01/2005	276	Oui

TAB. 3 – *Tables des livres (échantillon)*

Chaque ligne d'une table doit être identifiable de manière unique par une valeur qui n'est pas susceptible de changer au cours du temps. Dans notre exemple, les livres de la bibliothèque doivent être numérotés arbitrairement et sans doublon (un doublon, c'est deux fois la même valeur).

Dans la table des livres, il faut donc ajouter une colonne `numéro de livre` de type `NuméroAuto` (tableau 4). La numérotation des livres commence alors à 1 et s'incrémente automatiquement de 1 à chaque nouvelle ligne insérée dans la table[4].

numéro livre	titre	auteur(s)	date d'acquisition	nombre de pages	emprun-table
...	...	...	...	...	...

TAB. 4 – *Table des livres avec sa clé primaire*

La colonne qui numérote les lignes individuellement est appelée clé primaire de la table. Chaque table doit posséder sa clé primaire qui est définie à la création de la table à l'aide du bouton . Nous verrons par la suite, qu'une clé primaire peut être composée de plusieurs colonnes, mais qu'il faut limiter cette possibilité à des situations bien précises (liaisons plusieurs-à-plusieurs). Pour les tables simples, la clé primaire est simplement un numéro automatique.

Notons qu'Access autorise une clé primaire de type `Texte` (comme les numéros ISBN pour les livres ou les plaques d'immatriculation pour les voitures), mais pour des raisons de performances, ce n'est pas conseillé. Il vaut toujours mieux ajouter une colonne de type `NuméroAuto` qui fournit une numérotation, certes arbitraire, mais performante à tout point de vue.

D'ailleurs, un numéro de sécurité sociale, un code postal ou un numéro de téléphone sont également de mauvaises clés primaires car, rappelons-le, ce sont des chaînes de caractères, même si ces chaînes ne comportent que des chiffres. En effet, tous ces numéros peuvent commencer par un 0, ce qui n'est pas possible avec un entier en informatique. Il suffit donc de les stocker dans des colonnes de type `Texte`, mais qui ne sont pas clé primaire.

A contrario, les codes barres fournissent de bonnes clés primaires à condition qu'ils ne puissent ni commencer par 0 ni comporter un caractère autre que les chiffres 0, 1, 2, 3, 4, 5, 6, 7, 8 ou 9.

4. le numéro d'une ligne supprimée ne sera pas ré-attribué, mais ce n'est en aucun cas un problème

1.4 Contraintes

Les valeurs saisies dans la colonne `poids` évoquée dans la section 1.2 page 9, ne doivent pas être négatives ou nulles. Il faut donc ajouter la contrainte « valide si > 0 » à cette colonne, comme sur la figure 6. Grâce à cette contrainte, si une valeur négative ou nulle est saisie, Access la refuse et affiche le message d'erreur prévu à cet effet. Il s'agit d'une règle de validation de colonne.

Il en va de même pour les livres, dont le nombre de pages doit être strictement positif. Mais il faut reconnaître que le nombre de pages n'est pas une donnée vitale concernant les livres, elle peut recevoir la valeur vide dans certaines lignes.

Par contre, la colonne `titre` est une donnée essentielle pour un livre, il faut donc lui ajouter la contrainte `Null interdit` (autrement dit, valeur vide non autorisée), comme sur la figure 8. Par défaut, la valeur vide est toujours autorisée (autrement dit, `Null interdit` est positionné à `Non`).

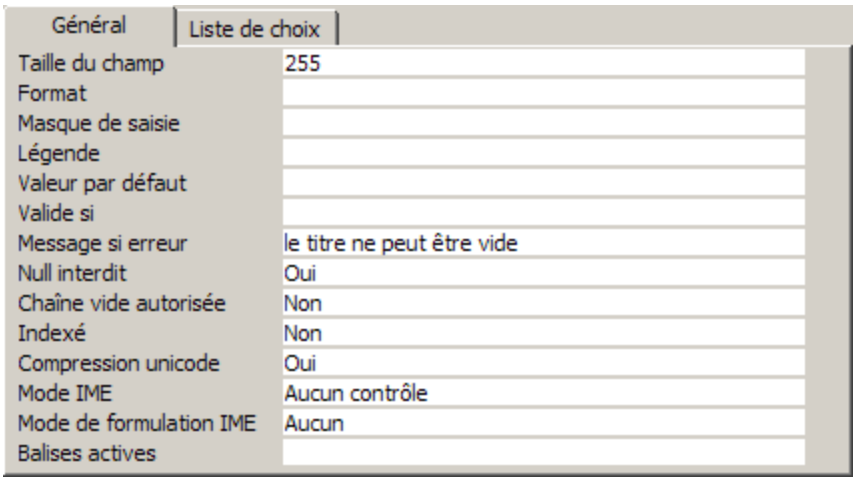


FIG. 8 – *Contraintes de non vacuité*

Ensuite, notons qu'une chaîne de caractères peut ne contenir aucun caractère. Dans ce cas la colonne n'est pas vide, mais la chaîne qu'elle contient est vide. Comme il ne faudrait pas que ce soit le cas pour un titre de livre, il convient d'ajouter, à cette même colonne `titre`, la contrainte `Chaîne vide autorisée = Non` (figure 8).

Ces premières règles de validation sont des contraintes de colonne, au même titre que les clés primaires de la section précédente, que l'unicité évoquée dans la section 2.6 page 23 et que les clés étrangères de la section suivante. Mais auparavant, évoquons les contraintes de tables.

Si l'énoncé d'une contrainte porte sur plusieurs colonnes de la table, alors il est possible de le saisir, non pas dans la contrainte `Valide si`, mais dans la contrainte `Valide si` de la table elle-même, en accédant aux propriétés de la table, à l'aide du bouton (accessible en mode création). Apparaît alors la fenêtre de la figure 9, dans laquelle nous pouvons saisir la règle de validation de la table, ainsi qu'un message d'erreur approprié.

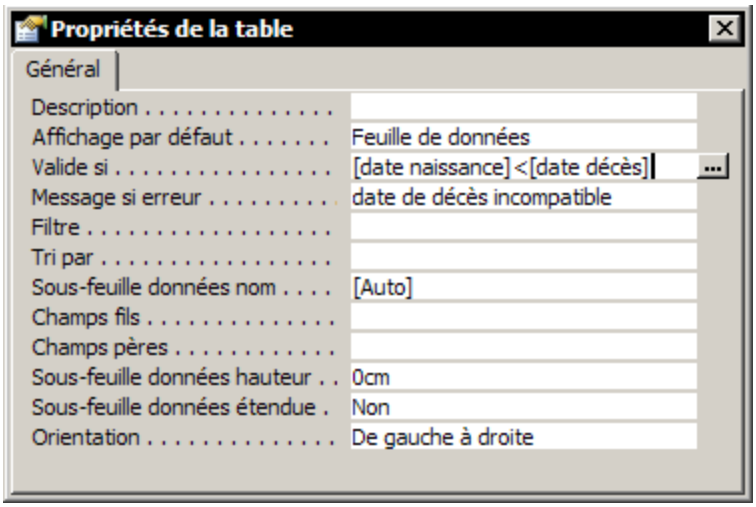


FIG. 9 – *Règle de validation de table*

Pour rédiger des règles de validation complexe, il ne faut pas hésiter à appeler le générateur d'expression (figure 10), à l'aide du bouton aux trois petits points , disponible en bout de rangée.

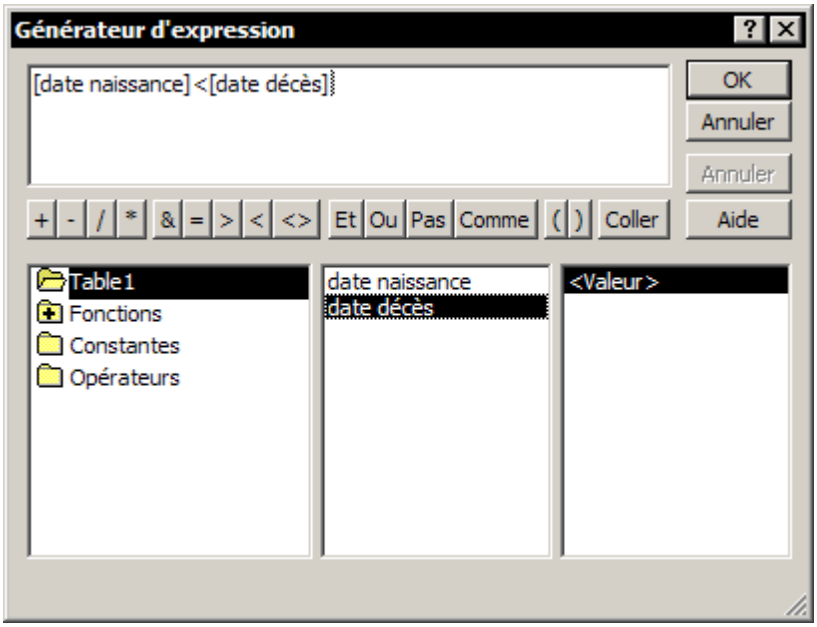


FIG. 10 – *Le générateur d'expression*

Par la suite, nous aurons l'occasion d'utiliser ce générateur d'expression à plusieurs reprises.

2 Clés étrangères

Les tables d'une base de données ne sont presque jamais isolées les unes des autres. Au contraire, il est presque toujours nécessaire d'établir un lien sémantique entre les lignes des différentes tables. C'est le rôle des clés étrangères.

Le travail de création des clés étrangères s'effectue *via* l'outil `Relations...` d'Access (figure 11) et ne doit surtout pas être négligé. En effet, une fois que toutes les clés étrangères sont parfaitement établies, beaucoup d'opérations deviennent automatiques quant à la création des requêtes, des formulaires et des états, grâce aux assistants.

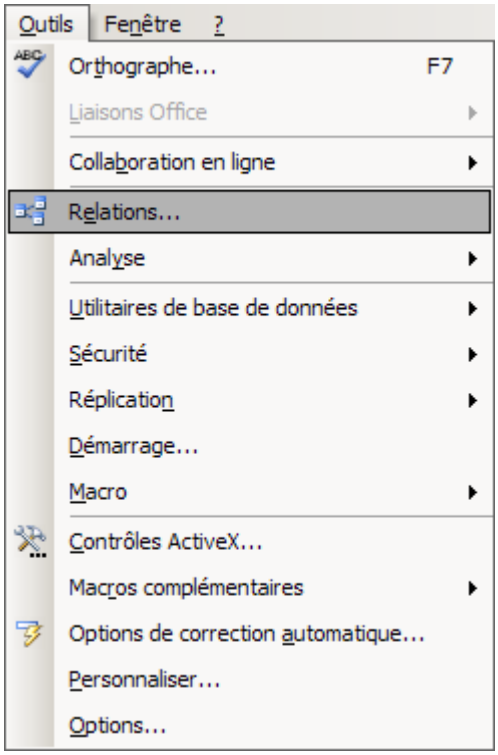


FIG. 11 – *Le menu le plus important d'Access, car il permet d'éditer le schéma relationnel de la base*

L'organisation des clés étrangères dans les tables d'une base de données, requiert au préalable une réflexion conceptuelle sur la sémantique des données. Nous supposons dans ce document, que ce travail de conception est déjà fait[5] et qu'il ne reste plus qu'à l'implémenter avec Access.

5. avec, de préférence, une analyse des dépendances fonctionnelles et l'élaboration d'un schéma entités-associations, comme décrit dans le document [Gruau] disponible sur Internet

2.1 Normalisation

La construction des tables dans une base de données doit obéir à un certain nombre de règles de bonne conduite. Au minimum, les tables doivent vérifier les trois premières formes normales.

Première Forme Normale : chaque cellule d'une table ne doit recevoir qu'une seule valeur (à un instant donné).

Contre-exemple : la colonne `auteur(s)` de la tables des livres (dont un échantillon est disponible dans le tableau 4 page 11) peut recevoir plusieurs valeurs, puisqu'un livre peut être écrit par plusieurs auteurs. La solution pour que la table `livres` respecte la première forme normale, consiste à supprimer la colonne `auteur(s)` au profit d'une table séparée pour les auteurs, sans oublier d'ajouter une colonne clé primaire à cette table (tableau 5). D'ailleurs, cette table séparée pour les auteurs permet d'éviter qu'un même auteur ne soit orthographié de deux manières différentes (à condition toutefois de prendre soin de ne pas le saisir deux fois dans cette table).

numéro auteur	auteur
1	Stephen King
2	Marie-France Etchegoin
3	Frédéric Lenoir

TAB. 5 – *Tables des auteurs (échantillon)*

La liaison (plusieurs-à-plusieurs) entre les livres et leurs auteurs est expliquée dans la section 2.3 et la deuxième forme normale est abordée à cette occasion. Mais auparavant, nous pouvons énoncer la troisième forme normale, qui ne nécessite pas la connaissance préalable des liaisons plusieurs-à-plusieurs.

Troisième Forme Normale : chaque colonne ne doit dépendre directement que de la clé primaire.

Un contre-exemple est donné par la table des avions (issue d'une base de données pour un aéroport) du tableau 6. Le problème est que le constructeur d'un avion et sa capacité ne dépendent pas directement du numéro d'avion, mais du type d'avion. Il y a donc redondance d'information, ce qui multiplie le travail de maintenance et induit un risque d'incohérence (l'avion numéro 3 est d'ailleurs un A320 qui possède Boeing comme constructeur).

numéro avion	nom	compagnie	type	constructeur	capacité
1	AF1244	Air France	A320	Airbus	240
2	BA7831	British Airways	B747	Boeing	310
3	KLM2034	KLM	A320	Boeing	240
4	AF6578	Air Frace	B747	Boeing	310

TAB. 6 – *Tables des avions (échantillon)*

Pour respecter la troisième forme normale, la solution consiste à retirer les colonnes problématiques (`type`, `constructeur` et `capacité`), au profit d'une table supplémentaire pour les types d'avion (tableau 7).

numéro type avion	type	constructeur	capacité
1	A320	Airbus	240
2	B747	Boeing	310

TAB. 7 – *Tables des type d'avions (échantillon)*

Dans le même ordre d'idée, dans la table des avions (tableau 6) le nom de la compagnie revient plusieurs fois et risque donc d'être mal orthographié (c'est le cas du deuxième avion d'Air France). C'est aussi une forme de redondance qu'il est préférable de remplacer par une table supplémentaire pour les compagnies, même si pour l'instant nous n'y stockons que leur nom (tableau 8).

numéro compagnie	nom
1	Air France
2	British Airways
3	KLM

TAB. 8 – *Table de codification des compagnies aériennes (échantillon)*

Bref, il ne faut pas hésiter à introduire de petites tables de codification supplémentaires afin d'éviter toute redondance et de réduire la taille des autres tables.

2.2 Liaison un-à-plusieurs

Dans la table des avions, il ne reste plus que quatre colonnes (tableau 9).

numéro avion	nom	numéro compagnie	numéro type avion
1	AF1244	1	1
2	BA7831	2	2
3	KLM2034	3	1
4	AF6578	1	2

TAB. 9 – *Tables des avions normalisée (échantillon)*

Les colonnes `numéro compagnie` et `numéro type avion` sont désormais codées : la colonne `numéro compagnie` contient la valeur correspondante de la clé primaire de la table des compagnies. La colonne `numéro compagnie` est une <u>clé étrangère</u> qui référence la table des compagnies. Idem pour la colonne `numéro type avion` qui référence la table des types d'avions.

Rappelons que les clés primaires étant de type `NuméroAuto`, les clés étrangères sont de type `Numérique`.

Pour créer la liaison entre deux tables (dans l'outil `Relations...`), il suffit de cliquer sur la colonne clé primaire, puis de glisser (sans lâcher le clic) le curseur illisible vers la clé étrangère voulue (ou l'inverse comme sur la figure 12).

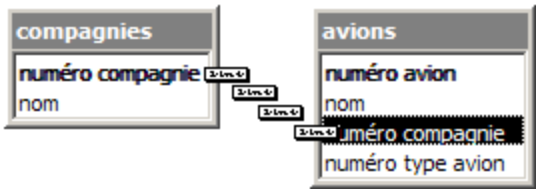


FIG. 12 – *Création d'une clé étrangère sur le schéma relationnel*

En représentant la liaison entre une clé étrangère et sa clé primaire par un connecteur (symbole ∞ côté clé étrangère et 1 côté clé primaire), nous pouvons organiser la description des colonnes des tables sous la forme d'un schéma relationnel (figure 13), sur lequel les clés primaires sont représentées en gras.

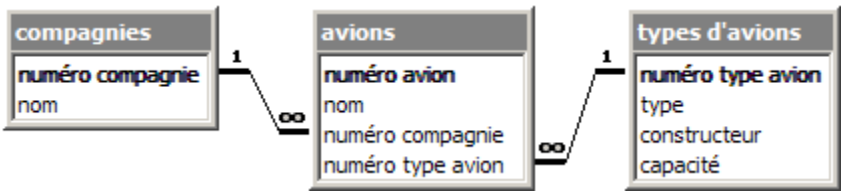


FIG. 13 – *Schéma relationnel entre les tables* `avions`, `compagnies` *et* `types d'avions`

Par ailleurs, une clé étrangère ne doit absolument pas contenir une valeur qui n'existe pas dans la clé primaire correspondante (sinon la référence est brisée). Lors de la création d'une clé étrangère sur le schéma relationnel, il faut donc préciser systématiquement à Access de maintenir l'intégrité référentielle[6] (figure 14) ;

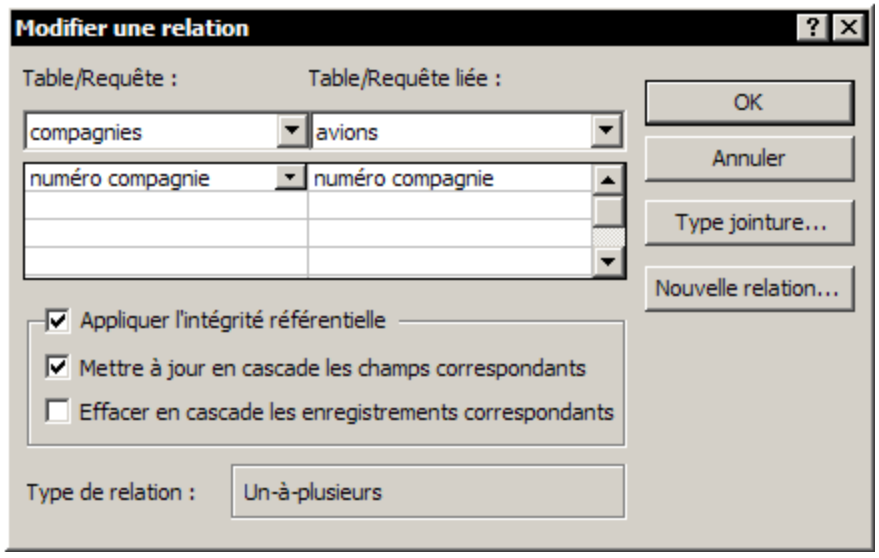


FIG. 14 – *Réglages conseillés pour l'intégrité référentielle (cette fenêtre est obtenue en double cliquant sur la liaison à éditer)*

6. cela implique généralement de supprimer la valeur par défaut (figure 15)

Les types de jointure (bouton `Type jointure...`) sont abordés dans la section 3.9 page 45, mais c'est le type 1 qui convient en général sur le schéma relationnel de la base[7].

Grâce à l'intégrité référentielle, une ligne dont la clé primaire est référencée par une clé étrangère ne peut pas être supprimée (cela évite que soit supprimé un type d'avion auquel est encore lié un avion).

À moins d'avoir précisé à Access qu'il fallait `Effacer en cascade les enregistrements correspondants`, ce qui n'est pas conseillé. Cette absence de suppression en cascade oblige cependant le développeur de la base de prévoir une programmation spécifique de certains boutons de suppression (comme dans la section 8.4.2 page 132).

Par ailleurs, une clé étrangère peut légitimement changer de valeur au cours du temps, lorsqu'elle est affectée à une autre valeur de clé primaire. Mais normalement, une clé primaire ne change pas de valeur. Ceci dit, dans le doute, il vaut mieux toujours dire à Access de `Mettre à jour en cascade les enregistrements correspondants` (comme sur la figure 14) ;

Enfin, il faut également se poser la question importante de savoir si la clé étrangère peut ou ne doit pas recevoir la valeur vide, auquel cas la contrainte doit être ajustée pour cette colonne (figure 15).

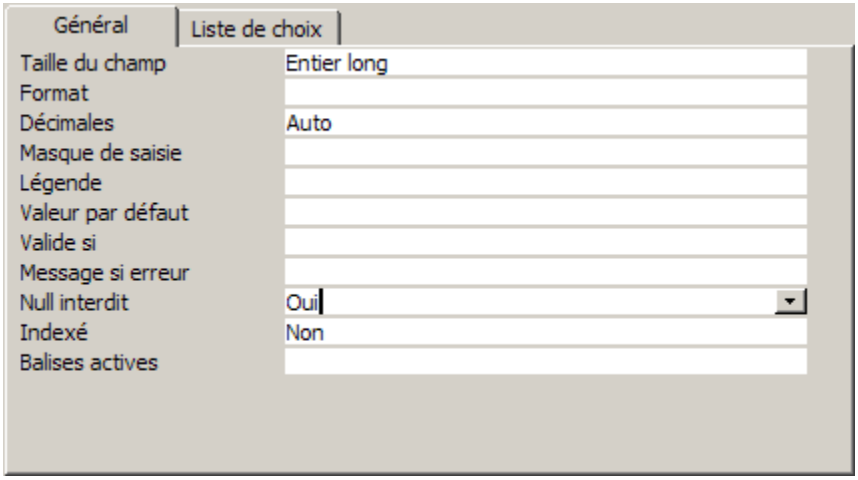


FIG. 15 – *Clé étrangère obligatoire (non vide)*

Il s'agit-là d'une contrainte forte, qui oblige l'utilisateur des formulaires à saisir d'abord les données de la table référencée, puis les données de la table qui possède la clé étrangère. Cela exige un peu de programmation VBA, comme dans la section 7.2.1 page 110).

7. ce qui n'est pas le cas sur le schéma relationnel d'une requête

2.3 Liaison plusieurs-à-plusieurs

À un avion correspond un type d'avion et à un type d'avion peuvent correspondre plusieurs avions. La liaison entre les tables `avions` et `types d'avions` est dite un-à-plusieurs. Chaque liaison un-à-plusieurs est gérée par une clé étrangère, comme dans la section précédente.

Par contre, un livre peut être écrit par plusieurs auteurs et un auteur peut écrire plusieurs livres. La liaison entre les tables `livres` et `auteurs` est dite plusieurs-à-plusieurs. Dans ce cas, une table de jonction supplémentaire est nécessaire. En l'occurrence, il s'agit de la tables des écritures (figure 16).

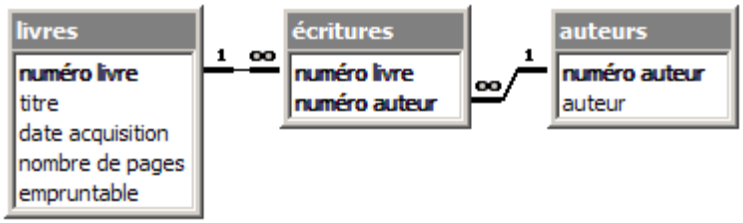


FIG. 16 – *Table de jonction pour une liaison plusieurs-à-plusieurs*

Dans la table de jonction `écritures`, nous avons deux clés étrangères qui compose la clé primaire de la table. La clé primaire est définie en premier, à la création de la table, en sélectionnant les deux colonnes qui la compose (à l'aide de la marge à gauche, où se situe le triangle blanc sur la figure 17) puis en cliquant sur le bouton clé primaire .

FIG. 17 – *Création d'une clé primaire sur deux colonnes pour une table de jonction*

Ensuite, les deux clés étrangères sont créées successivement dans la fenêtre des relations, en suivant la même procédure que pour la clé étrangère de la figure 12 page 17).

Remarques :
- une clé primaire peut être composée de plusieurs colonnes, mais nous limitons ce cas de figure aux liaisons plusieurs-à-plusieurs ;
- une colonne peut à la fois être clé étrangère et appartenir à une clé primaire ;
- les clés étrangères d'une table de jonction ne doivent pas accepter la valeur vide et il faut généralement qu'Access applique l'intégrité référentielle (avec éventuellement mise à jour en cascade, mais pas suppression en cascade), comme sur la figure 14.

Pour lier les auteurs des deux livres donnés dans le tableau 3 page 11, il faut trois lignes dans la table `écritures`, car le deuxième livre a été écrit par deux auteurs (tableau 10).

numéro livre	numéro auteur
1	1
2	2
2	3

TAB. 10 – *Tables des écritures (échantillon)*

2.4 Deuxième forme normale

Un autre exemple de table de jonction nous est donné par la liaison entre la tables des films et la table des acteurs. En effet, un film peut être interprété par plusieurs acteurs et un acteur peut jouer dans plusieurs films. Il faut donc une table de jonction pour les interprétations, entre les tables `films` et `acteurs`. Dans cette table, nous pouvons ajouter une colonne pour stocker le nom du personnage (figure 18).

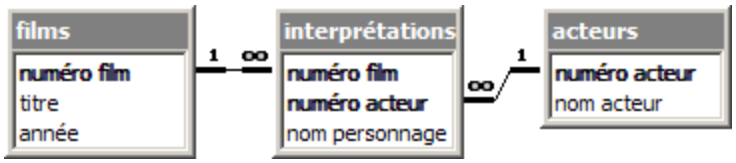


FIG. 18 – *Autre colonne dans une table de jonction*

Deuxième forme Normale : une clé primaire peut être composée de plusieurs colonnes, mais les autres colonnes de la table doivent dépendre de la clé primaire en entier (et non pas, d'une partie de la clé primaire).

Si la colonne `année` avait été placée dans la table `interprétations`, alors la deuxième forme normale n'aurait pas été vérifiée, car l'année ne dépend que du numéro de film et pas du numéro d'acteur. Bref, les colonnes doivent aller dans la table dont la clé primaire leur correspond parfaitement (si aucune clé primaire ne correspond, alors il faut ajouter une table).

Malheureusement, dans certains films, un acteur peut interpréter plusieurs personnages. Autrement dit, étant donné un numéro de film et un numéro d'acteur, nous pouvons avoir plusieurs noms de personnages. La colonne `nom personnage` ne vérifie donc pas la deuxième forme normale.

Une solution envisageable consisterait à inclure la colonne `nom personnage` dans la clé primaire de la table `interprétations`. Mais, rappelons qu'il n'est pas conseillé d'inclure une chaîne de caractères dans une clé primaire. C'est pourquoi nous proposons plutôt d'ajouter une numérotation arbitraire et automatique à la table `interprétations` (figure 19).

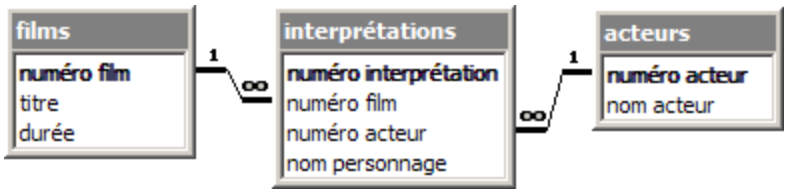
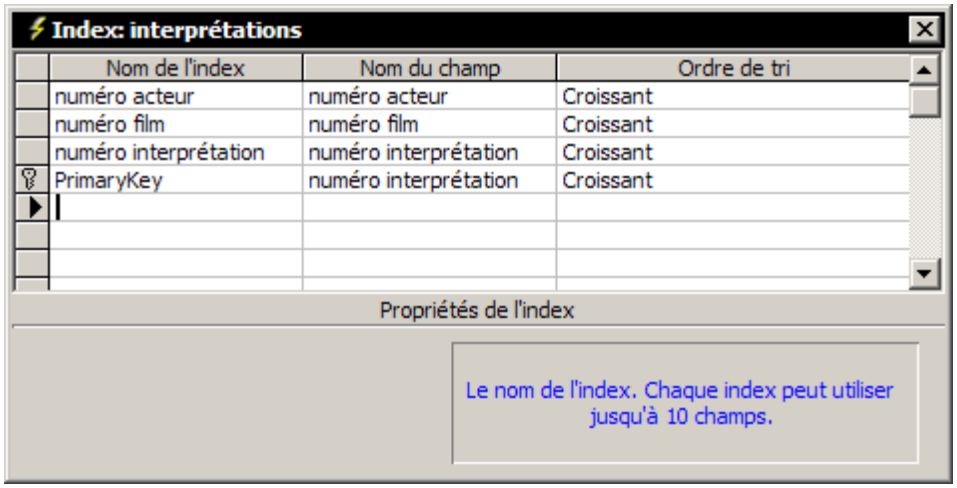


FIG. 19 – *Application de la deuxième forme normale à une table de jonction*

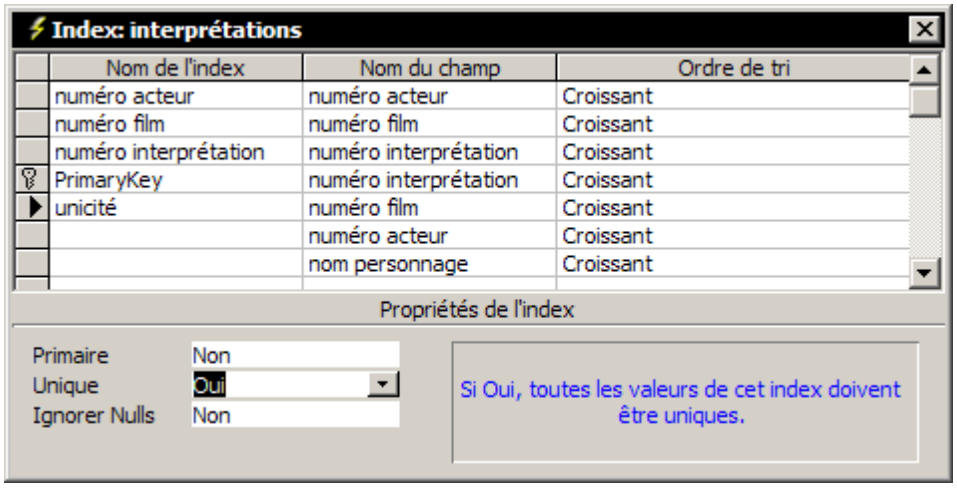
Le seul inconvénient de cette technique est la possibilité pour la table `interprétations` de posséder plusieurs fois la même ligne (le même film, le même acteur et le même personnage, mais pas la même valeur de clé primaire). Cet inconvénient peut être levé en ajoutant un index composite, fondé sur toutes les colonnes de la table (sauf la clé primaire).

Pour prévenir ce problème, nous pouvons ajouter une contrainte d'unicité commune pour les trois colonnes `numéro film`, `numéro acteur` et `nom personnage`. Cette contrainte d'unicité correspond à un index sans doublon qui est composite (c'est-à-dire qui porte sur plusieurs colonnes à la fois).

Pour créer un index composite, il suffit de cliquer sur le bouton (disponible en mode création) pour afficher la fenêtre de la figure 20(a), dans laquelle nous observons déjà la présence de la clé primaire, de son index, ainsi que les index des deux clés étrangères.



(a) les index non composites



(b) ajout de l'index non composite, sans doublon

FIG. 20 – *Intervention dans la fenêtre des index*

Nous ajoutons manuellement, un index nommé `unicité`, dont nous ne répétons pas le nom dans les deux rangées qui suivent afin de bien préciser à Access qu'il s'agit d'un index portant sur trois colonnes. Pour assurer l'absence de doublons, nous positionnons simplement à `Oui` la propriété `Unique`. Il ne reste plus alors qu'à sauvegarder la table.

2.5 Gestion des dates et du caractère historique

De retour dans notre bibliothèque, nous ajoutons une table pour les membres et nous voulons stocker à la fois les emprunts en cours et les emprunts historiques.

Pour les emprunts en cours il suffit de connaître, pour chaque livre :
- le numéro du membre qui l'emprunte (cette colonne peut être vide, ce qui signifie que le livre n'est pas emprunté actuellement) ;
- la date d'emprunt et la date de retour prévu.

Comme ces trois informations dépendent directement du `numéro livre`, nous les ajoutons à la table des livres (figure 21).

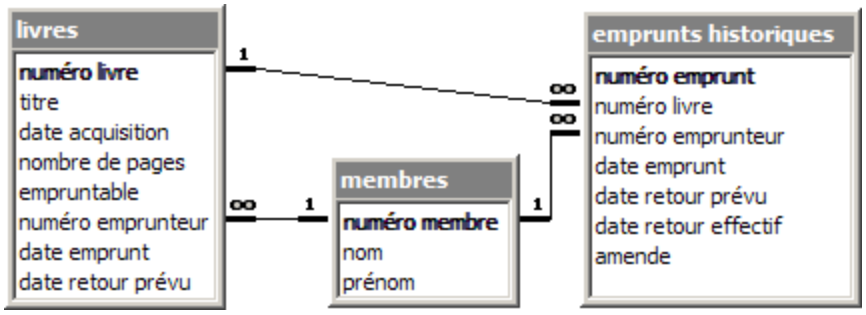


FIG. 21 – *Table historique*

Par contre, un livre peut faire l'objet de plusieurs emprunts au cours du temps. Dans ces conditions, nous ajoutons la table `emprunts historiques`, dans laquelle nous pouvons en plus stocker la date de retour effectif et le montant de l'amende éventuelle en cas de retard.

Pour la clé primaire de cette table `emprunts historiques`, nous appliquons la deuxième forme normale qui ne nous permet de pas de nous contenter d'une clé primaire composées du numéro de livre, du numéro de membre et de la date d'emprunt, car un membre peut emprunter (puis rendre) un livre plusieurs fois le même jour. Nous choisissons donc une numérotation arbitraire et automatique.

Dans tous les cas, il n'est pas conseillé de faire entrer une date dans la clé primaire d'une table. Nous pourrions mettre toutes les dates utiles dans une table séparée (liée ensuite par clé numérique), mais ce n'est pas approprié non plus, tant qu'aucune information relative aux dates (comme le caractère férié, par exemple), n'a besoin d'être stockée.

2.6 Liaison un-à-un

Dans une base de données pour une entreprise, sont stockés : la liste des services qui composent l'entreprise et la liste des employés de l'entreprise. Comme un service est dirigé par un employé et comme un employé ne peut diriger qu'un service (au maximum), cette liaison entre les tables `services` et `employés` est dite un-à-un.

Une liaison un-à-un est en réalité une liaison un-à-plusieurs particulière, dans laquelle la clé étrangère reçoit en plus une contrainte d'unicité (doublons non autorisés). Dans la table des services, nous avons la clé étrangère `numéro directeur` pour identifier l'employé qui dirige le service, mais pour assurer le fait qu'un employé ne peut pas diriger plus d'un service, il suffit de préciser pour cette colonne `numéro directeur` que les doublons ne sont pas autorisés.

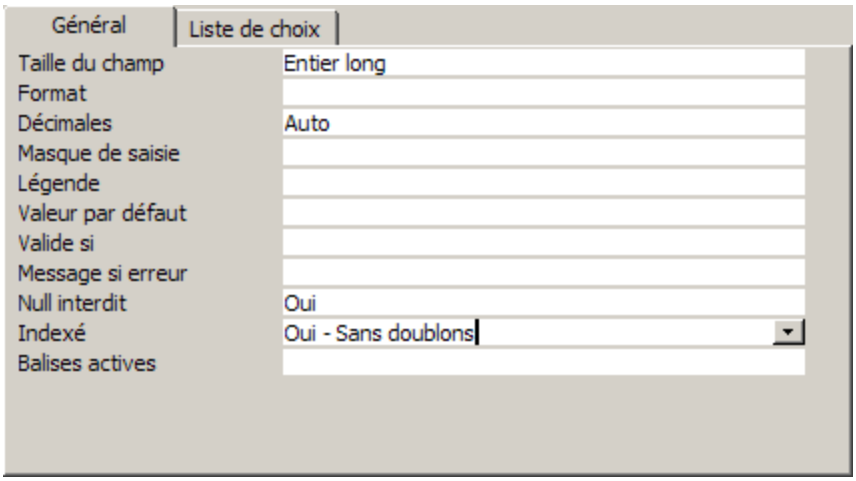


FIG. 22 – *Interdiction des doublons pour une liaison un-à-un*

Cette opération s'effectue dans la table `services` en mode création en ajustant les propriétés du champ `numéro directeur` : la propriété `Indexé` qui au départ vaut `Oui - Avec doublons` doit être positionnée à `Oui - Sans doublons` (figure 22). Sur le schéma relationnel, le symbole ∞ côté clé étrangère disparaît au profit du symbole 1 qui indique que la liaison est de type un-à-un (figure 23).

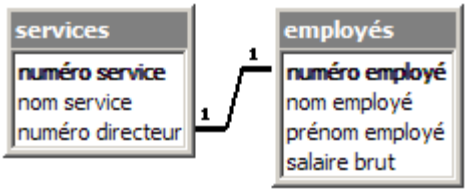


FIG. 23 – *Liaison un-à-un*

C'est la première fois que nous rencontrons une colonne, qui n'est pas clé primaire, recevoir une contrainte d'unicité : il s'agit d'une colonne clé étrangère (donc de type numérique) en liaison un-à-un. Il se trouve qu'une colonne possédant une contrainte d'unicité, c'est-a-dire possédant un index sans doublons, peut être référencée par une clé étrangère, au même titre qu'une clé primaire (figure 24).

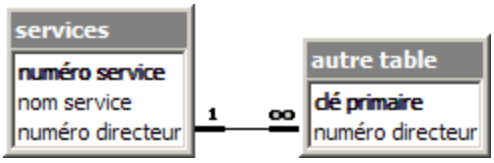


FIG. 24 – *Une clé étrangère qui ne référence pas une clé primaire (exemple théorique)*

D'ailleurs, pour des raisons de performances, il est conseillé de n'indexer que les colonnes de type `Numérique`, comme les clés primaires et les clés étrangères. Donc en dehors des clés primaires, une liaison un-à-un est pratiquement le seul cas où une contrainte d'unicité devrait apparaître.

De plus, dans notre exemple, il est préférable de faire pointer une clé étrangère vers le `numéro service`, plutôt que vers le `numéro directeur` qui est susceptible de changer au cours du temps. La possibilité de référencer une colonne non clé primaire est donc inexploitée en général.

Cependant, Dans la section 9.3 page 146, dédiée à la création d'un formulaire de connexion, nous rencontrerons cependant une table dans laquelle une colonne de type `Texte` est indexée sans doublon, afin d'assurer l'unicité du nom d'utilisateur.

Cette contrainte d'unicité s'ajoute à la liste des contraintes énoncée dans la section 1.4 page 12. Elle permet également d'implémenter, en relationnel, une notion empruntée à la modélisation objet (section suivante).

2.7 Héritage

Les liaisons un-à-un permettent également d'implémenter la notion d'héritage. La notion d'héritage consiste à factoriser au sein d'une table mère, les colonnes communes à plusieurs tables filles, en laissant dans les tables filles les colonnes spécifiques qui permettent de les différencier (figure 25).

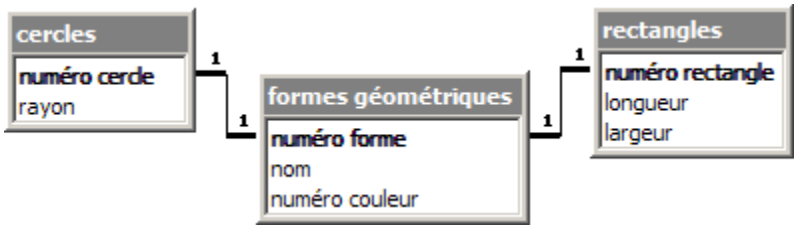


FIG. 25 – *Deux tables filles qui héritent de leur table mère*

La présence de la table `formes géométriques` permet de changer la couleur[8] des tous les cercles et de tous les rectangles en une seule requête. L'intérêt de cet exemple est limité, mais nous pouvons imaginer que de nombreuses autres tables peuvent hériter de la table `formes géométriques` (les triangles, les quadrilatères, etc.).

La notion d'héritage peut être étendue[9] sur plusieurs générations, ce qui démultiplie les possibilités de hiérarchisation des tables. Dans tous les cas, le lien d'héritage entre une table fille et une table mère correspond à une formulation « est un » :

- un cercle « est une » forme géométrique ;
- un rectangle « est une » forme géométrique.

Toutefois, il ne faut pas voir d'héritage à chaque fois que nous pouvons dire « est un », car il faut en plus que la table mère <u>ne</u> possède <u>que</u> les colonnes communes de ses tables filles. Par exemple, un cercle « est <u>mathématiquement</u> un » ovale. Pourtant, la table `cercles` ne peut pas être une table fille de la table `ovales` car cette dernière possède davantage de colonnes : un ovale possède notamment deux rayons (un rayon intérieur et un rayon extérieur).

En pratique dans une base de données, l'héritage est surtout utilisé pour séparer une table ayant beaucoup de colonnes en deux tables (ou davantage) : une table avec les colonnes obligatoires et une table avec les colonnes facultatives.

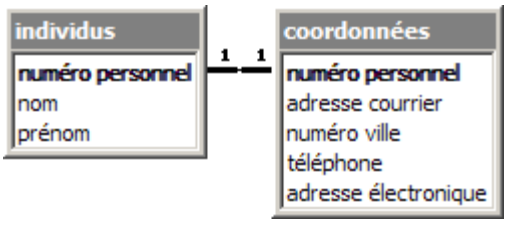


FIG. 26 – *Utilisation de l'héritage pour séparer les colonnes principales des colonnes complémentaires*

Par exemple, dans une table qui liste des individus, nous ne connaissons pas forcément les coordonnées de tous les individus. Plutôt que d'ajouter, dans la table `individus`, des colonnes qui seront vides dans certaines lignes, nous préférons créer une deuxième table séparée pour les colonnes des coordonnées (figure 26). Pour attribuer les bonnes coordonnées au bon individu, nous adoptons, comme clé primaire de cette nouvelle table, une clé étrangère[10] vers la clé primaire de la première table.

8. qui est en fait une clé étrangère vers la table des couleurs qui n'est pas représentée sur la figure 25
9. la notion d'héritage multiple correspond à celle de table de jonction
10. dont l'homonymie n'est pas obligatoire

2.8 Auto-liaison

Une clé étrangère peut être dirigée vers la clé primaire de la même table, auquel cas nous parlons d'auto-liaison. Si dans la table des employés nous ajoutons pour chaque employé, le numéro de son supérieur hiérarchique, cette nouvelle colonne est une clé étrangère vers la clé primaire de la table des employés.

Pour réaliser cette auto-liaison dans l'outil `Relations...` d'Access, il faut afficher deux fois la table des employés (avec le bouton qu'il faut invoquer lorsqu'une table a été oubliée sur le schéma), ce qui permet de tirer, à la souris, la liaison entre la clé étrangère et la clé primaire de la copie (figure 27).

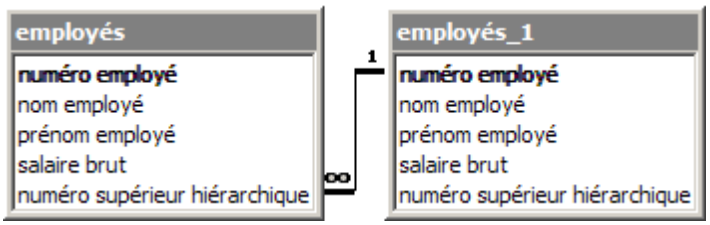


FIG. 27 – *Auto-liaison un-à-plusieurs*

Remarque : pour l'employé qui se trouve au sommet de la hiérarchie, sa colonne `numéro supérieur hiérarchique` est soit vide (et donc, vide autorisé pour cette table), soit le numéro de l'employé lui-même (ce qui permet d'interdire la valeur `Null`).

Un autre exemple d'auto-liaisons nous est donné par une base de données généalogique (figure 28), dans laquelle la table des personnes contient une clé primaire vers la mère (légale) et une clé primaire vers le père (légal).

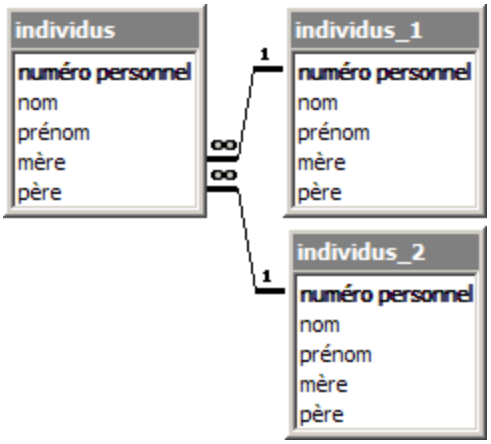


FIG. 28 – *Deux auto-liaisons et donc deux duplicata pour la table de départ*

Il n'est pas nécessaire d'ajouter quatres colonnes pour les grands-parents pour établir la liaison entre les parents et les grands-parents, car le travail effectué sur la figure 28 suffit, par récurrence.

2.9 Liaison composite

Une clé étrangère peut référencer une clé primaire composée de plusieurs colonnes, auquel cas, la clé étrangère est elle-même composite. C'est le cas sur l'exemple de la figure 29, où la liaison plusieurs-à-plusieurs entre les tables `parieurs` et `participations` génère une table `paris` dont la deuxième clé étrangère est composée de deux colonnes, puisque la table `participations` est elle-même une table de jonction issue d'une liaison plusieurs-à-plusieurs.

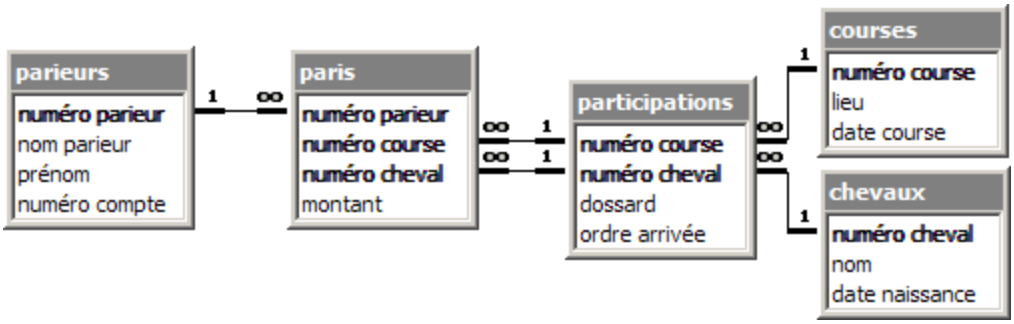


FIG. 29 – *Clé étrangère composite*

Contrairement à ce que nous pourrions croire en regardant la figure 29, il n'y a qu'une liaison entre les tables `paris` et `participations`, mais elle porte sur deux colonnes (la symbolique adoptée par Access est malheureusement ambiguë dans cette configuration).

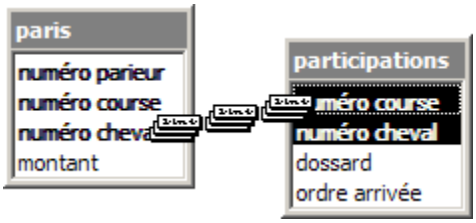


FIG. 30 – *Création d'une clé étrangère composite sur le schéma relationnel*

Par ailleurs, pour tirer la liaison composite entre ces deux tables, il faut sélectionner d'abord les deux colonnes côté clé primaire (figure 30) puis tirer le curseur illisible (sans lâcher la souris) vers la table côté clé étrangère (et pas l'inverse, contrairement à une liaison non composite). De plus, il faut préciser manuellement les colonnes de la clé étrangère dans la boîte de dialogue qui s'ouvre alors, car Access ne le fait pas automatiquement.

2.10 Sous-numérotation

Pour s'assurer de l'intégrité des données il est parfois indispensable de recourir à l'emploi d'une clé primaire partiellement étrangère, ce qui constitue encore un nouveau cas de figure. Cette situation est illustrée sur la figure 31 par les tables `trous` et `parties`.

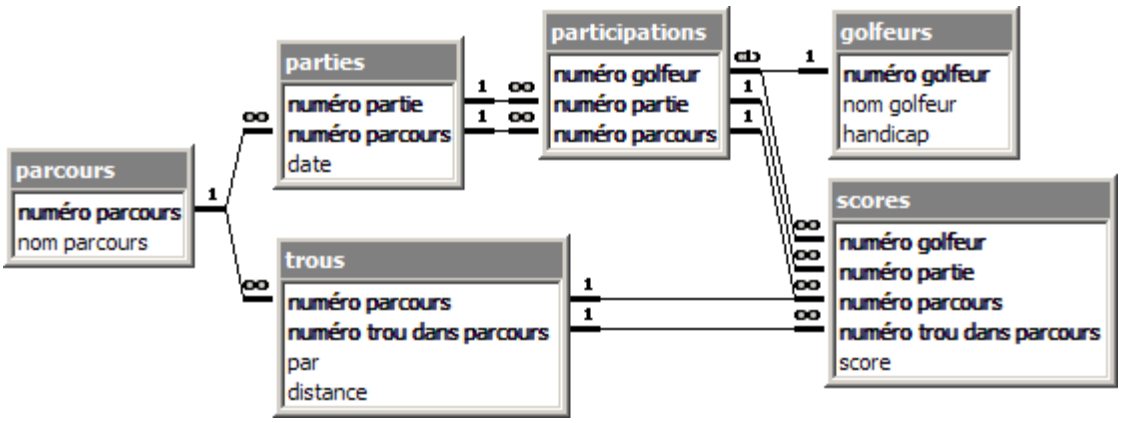


FIG. 31 – *Utilisation de clés primaires partiellement étrangères*

Les clés primaires des tables `trous` et `parties` sont composées de deux colonnes :

- une clé étrangère, `numéro parcours` ;
- et une sous-numérotation interne pour les trous qui ont le même numéro de parcours (idem pour les parties).

Cette sous-numérotation n'est pas de type `NuméroAuto`, mais de type `Numérique` manuel, car elle recommence à 1 quand nous changeons de numéro de parcours (généralement, les trous d'un parcours sont numérotés de 1 à 18).

Cette sous-numérotation se répercute dans les clés primaires des tables `participations` et `scores`. En conséquence de quoi, la première clé étrangère de la table `scores` est composées de trois colonnes : les trois colonnes qui composent la clé primaire de la table `participations`. Notons également, que la deuxième clé étrangère de la table `scores` ré-utilise astucieusement le `numéro parcours` de la première clé étrangère[11].

Sans ce dispositif basé sur la sous-numérotation, il pourrait y avoir un score dans la table `scores`, pour un trou qui n'appartient pas au parcours sur lequel la partie se joue. Il s'agit précisément de ce qui ne doit pas être permis par l'intégrité référentielle.

11. cet exemple représente ce que se fait de plus élaboré en matière d'intégrité référentielle et réclame donc un examen attentif de la part du lecteur

2.11 Conclusion

Comme nous le voyons sur l'exemple précédent, la création d'un schéma relationnel complet est une tâche ardue qui exige la plus grande rigueur. Le lecteur est donc invité à adopter une méthodologie de type Merise, basée sur l'étude préalable des dépendances fonctionnelles directes et la construction d'un schéma entités-associations normalisé, traduisible ensuite, de manière automatique, en un schéma relationnel correct. Pour cela, le lecteur est renvoyé à la référence [Gruau].

La lecture de cette méthodologie est également l'occasion de comprendre dans quelles situations une table de jonction entre trois tables[12] est légitime (comme sur la figure 32).

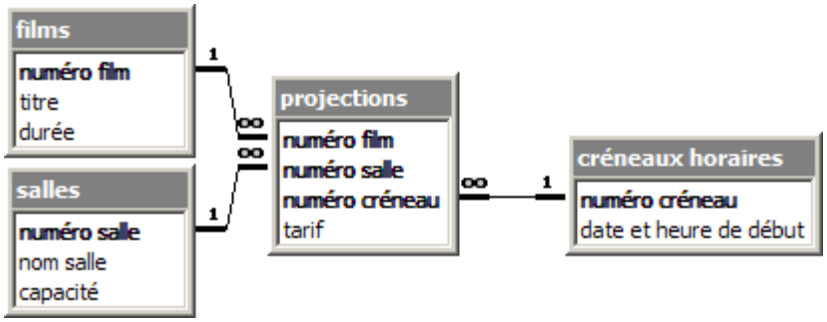


FIG. 32 – *Table de jonction ternaire*

La table de jonction `projections` est issue d'une . La seule possibilité de s'assurer de la légitimité d'une telle table passe par l'étude des dépendances fonctionnelles directes.

Une table de jonction entre quatre tables ou plus est parfois nécessaire, mais en pratique, nous rencontrons rarement cette configuration. En tout état de cause, pour ces tables de jonction aussi, l'études des dépendances fonctionnelles directes est indispensable.

12. une telle table comporte une clé primaire composée de trois clés étrangères, elle est donc entourée d'au moins trois symbole ∞

3 Requêtes de sélection

Une fois que les données sont bien organisées, elles peuvent être manipulées et exploitées par l'utilisateur de la base, *via* des formulaires et des états. Généralement, chaque formulaire et chaque état ne travaille pas sur l'ensemble des données, mais seulement sur une partie des données. Le rôle des requêtes de sélection est précisément de sélectionner les données pertinentes pour telle ou telle application.

Écrire une requête de sélection dans Access consiste à :
- choisir la ou les table(s) concernées ;
- lier ces tables convenablement (les clés étrangères sont une aide précieuse, mais qui ne suffit pas toujours) ;
- choisir la ou les colonne(s) (appelées champs par Access) concernées (dans les tables déjà choisies) ;
- préciser, pour chaque colonne sélectionnée, si elle doit ou non être affichée[13] ;
- donner éventuellement un ou plusieurs critère(s) pour ne conserver que certaines lignes des tables (il s'agit de condition(s) de sélection) ;
- dire si le résultat doit être trié et comment.

Dans ce document, nous notons ces informations sous la forme d'un tableau qui suit la même logique que l'outil visuel d'Access qui permet de créer une requête de sélection en `Mode création` (figure 33).

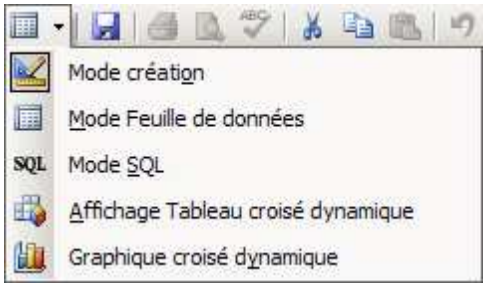


FIG. 33 – *Les différents modes pour les requêtes*

Nous abordons également l'écriture des requêtes simples en `Mode SQL` (Structured Query Language) et nous laissons au lecteur le soin de consulter le résultat en mode `Feuille de données`, `Tableau croisé dynamique` ou `Graphique croisé dynamique`, dont des exemples sont disponibles dans le document [Isoz] téléchargeable sur Internet.

13. les colonnes qui ne servent que de critère ou que de tri ne sont pas toujours affichées

3.1 Sélection simple

Pour afficher les livres empruntables avec leur titre, leur date d'acquisition et leur nombre de pages dans l'ordre alphabétique des titres, il suffit de réaliser une requête de sélection qui pioche les données dans une seule table (tableau 11).

les tables et leurs liaisons	`livres`
les colonnes et leur affichage	`titre`, `date d'acquisition`, `nombre de pages`, `empruntable` (non affichée)
les critères sur les lignes	`empruntable = Oui`
les tris et leur sens	`titre` (croissant)

TAB. 11 – *Requête de sélection mono-table*

En mode création, l'équivalent graphique pour cette requête est capturé sur la figure 34.

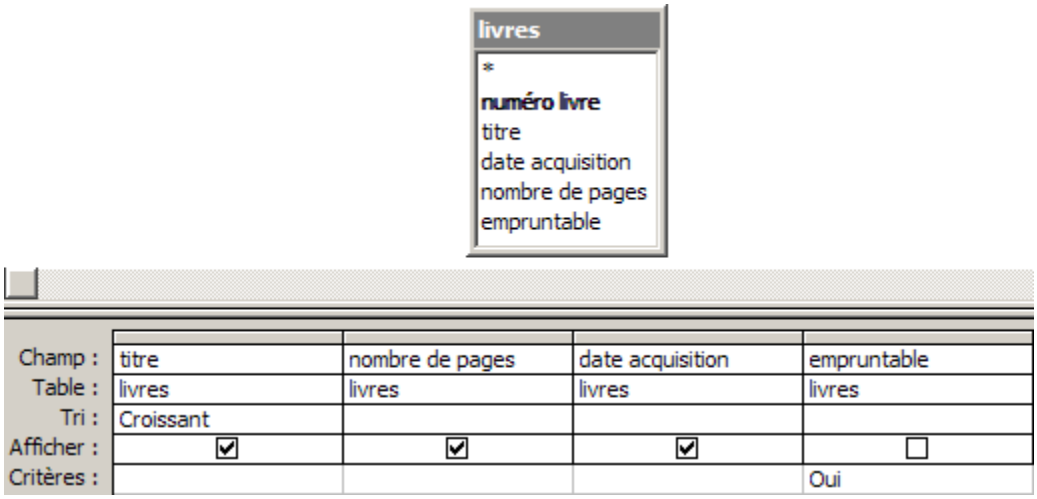


FIG. 34 – *Requête nommée* `livres_empruntables` *et utilisée par la suite*

La même requête peur également être écrite en SQL, auquel cas :
- les tables concernées sont précisées dans la clause `FROM` ;
- les colonnes à afficher dans la clause `SELECT` (entourées de crochets si leur nom comporte des espaces) ;
- les conditions de sélection dans la clause `WHERE` ;
- et les tris dans la clause `ORDER BY`.

La requête SQL de notre exemple précédent est donc la suivante :

```
1 SELECT titre, [date d'acquisition], [nombre de pages]
2 FROM livres
3 WHERE empruntable = Yes
4 ORDER BY titre ASC ;
```

La syntaxe SQL est purement anglophone, il faut donc traduire en anglais tous les mots-clés français autorisé par Access (comme `Oui` et `Non`, par exemple).

Pour afficher toutes les colonnes d'une table sans avoir à les préciser toutes, nous pouvons utiliser l'astérisque qui figure en premier dans la représentation de la table sur le schéma de la requête (figure 34). Par exemple, pour afficher tous les renseignements de tous les employés, la requête du tableau 12 est très simple.

les tables et leurs liaisons	`employés`
les colonnes et leur affichage	`*` (toutes les colonnes)
les critères sur les lignes	aucun (toutes les lignes)
les tris et leur sens	aucun (dans le désordre)

TAB. 12 – *Utilisation de l'astérisque dans une requête*

En l'absence de critère sur les lignes, toutes les lignes sont affichées et en l'absence de tri, les lignes sont affichées dans le désordre.

3.2 Propriétés de requête

Pour afficher la liste de propriétés d'une requête (figure 35), il suffit de cliquer sur le bouton ou de double-cliquer sur un endroit vide de l'espace réservé à son schéma relationnel.

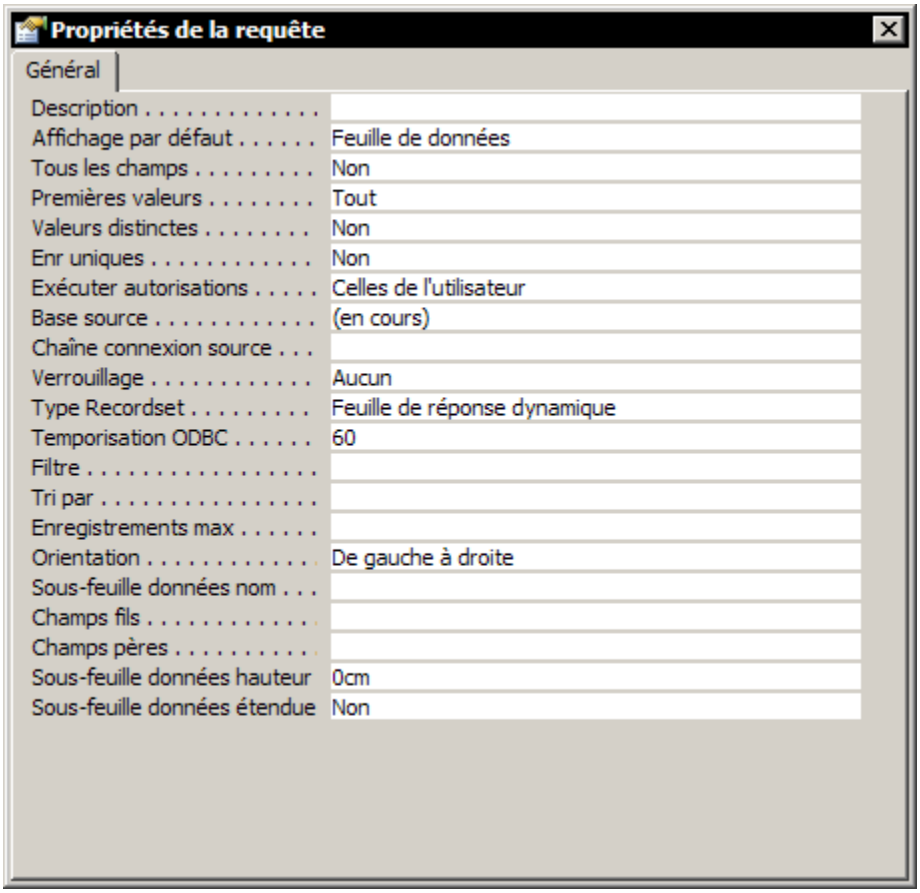


FIG. 35 – *La feuille des propriétés d'une requête*

Parmi toutes ces propriétés, seules deux retiennent notre attention dans ce document. Premièrement, la propriété `Valeurs distinctes` permet, si elle est positionnée à `Oui` (ce qui n'est pas le cas par défaut), de supprimer les doublons du résultat. Une requête peut en effet afficher plusieurs fois la même ligne[14], ce qui est souvent inutile.

Deuxièmement, la propriété `Premières valeurs` permet de limiter l'affichage du résultats aux premières lignes générées par la requête. Cette fonctionnalité, qui est également disponible à travers la liste déroulante de la figure 36, est utile pendant la phase de développement d'une requête excessivement longue à répondre.

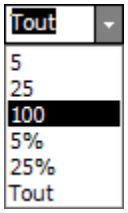


Fig. 36 – *Limitation de l'affichage du résultat aux premières lignes*

En SQL, ces deux propriétés se traduisent respectivement par l'ajout des mots-clés `DISTINCT` et `TOP` dans la clause `SELECT` :

```
1 SELECT DISTINCT TOP 100 titre, [date d'acquisition], [nombre de pages]
2 FROM livres
3 WHERE empruntable = Yes
4 ORDER BY titre ASC ;
```

Il convient cependant de ne plus limiter le nombre de lignes du résultat lorsque la requête est prête à être utilisée par un formulaire.

14. contrairement à une table, qui possède sa clé primaire, colonne sans doublon

3.3 Sélection sur plusieurs tables

Contrairement aux requêtes mono-tables précédentes, pour afficher les avions d'`Air France` avec leur nom, leur type et leur capacité, par constructeur décroissant, la requête doit faire appel à trois tables :

- la table des avions, pour afficher leur nom ;
- la table des types d'avions, pour afficher le type et la capacité, ainsi que pour pouvoir effectuer un tri sur le constructeur ;
- la table des compagnies, afin de sélectionner `Air France`.

les tables et leurs liaisons	`avions`, `types d'avions`, `compagnies` (ces tables sont correctement liées par le schéma relationnel)
les colonnes et leur affichage	`avions.nom`, `type`, `capacité`, `constructeur` (non affiché), `compagnie.nom` (non affichée)
les critères sur les lignes	`compagnie.nom = 'Air France'`
les tris et leur sens	`constructeur` (décroissant)

TAB. 13 – *Requête de sélection sur plusieurs tables*

Lorsque le nom d'une colonne est ambigu (c'est le cas de la colonne `nom` sur l'exemple du tableau 13), il faut lever l'ambiguïté en le préfixant par le nom de la table à laquelle la colonne appartient.

Dans une requête impliquant plusieurs tables, aucune table ne doit être isolée : elles doivent toutes être liées entre elles[15]. En conséquence, même si la requête n'avait pas eu à afficher le nom des avions, il aurait fallut faire appel à la table des avions, car c'est la seule qui permette d'effectuer le lien entre la tables des compagnies et celle des types d'avions. Le fait que les tables impliquées soient correctement liées par le schéma relationnel de la base est visible sur le schéma relationnel de la requête[16] (figure 37).

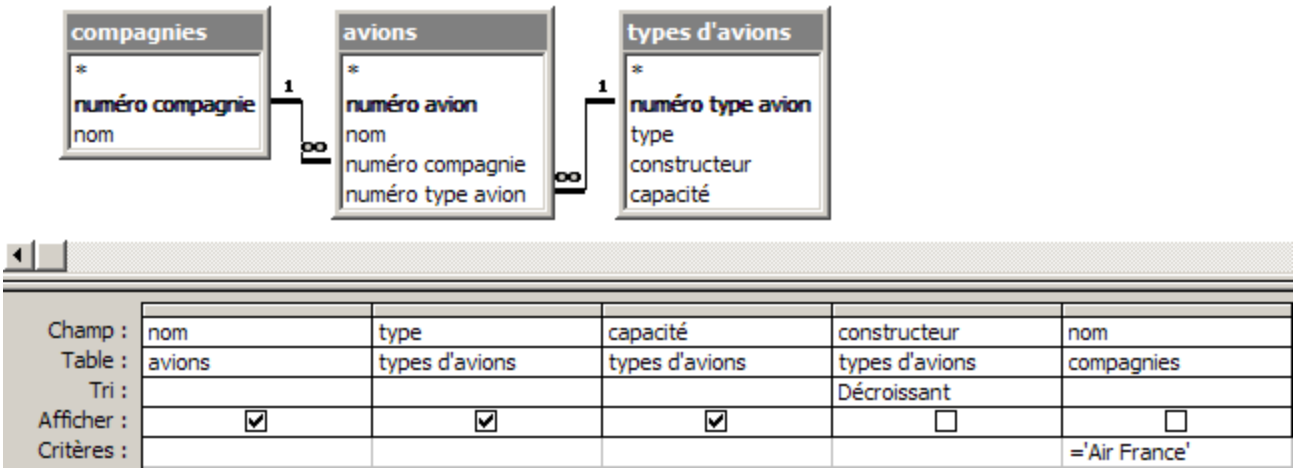


FIG. 37 – *Requête impliquant plusieurs tables*

La traduction SQL de la requête précédente fait apparaître dans la clause `FROM` le rappel des liaisons clés étrangères - clés primaires sous forme de conditions de jointure (opérateur `INNER JOIN ... ON`), dont la syntaxe Access est malheureusement trop contraignante. Nous nous contentons donc, dans ce document, d'écrire en SQL uniquement des requêtes mono-tables, réservant la création de requête sur plusieurs tables au mode création. Nous verrons que cela suffit, y compris à la programmation SQL dans VBA (à laquelle la section 8 est dédiée), grâce à la notion de vue (section suivante).

15. sans liaison, autrement dit, sans jointure entre deux tables, Access répond par le produit cartésien des deux tables, ce qui est rarement une bonne idée

16. nous verrons plus loin des schémas de requêtes qui diffèrent du schéma de la base

3.4 Vues

Une requête Access est systématiquement enregistrée et porte un nom. Seule sa programmation est enregistrée, son résultat est recalculé à chaque fois qu'elle est ouverte en `Mode Feuille de données`, notamment. L'avantage d'enregistrer une requête est que son résultat peut être utilisé comme une table dans une autre requête.

3.4.1 Ré-utilisation d'une requête dans une autre requête

Par exemple, pour afficher les titres des livres empruntables qui ont 100 pages ou moins (tableau 14), nous pouvons utiliser la requête `livres_empruntables` écrite précédemment, pour économiser la ré-écriture du critère sur le caractère empruntable.

les tables et leurs liaisons	`livres_empruntables`
les colonnes et leur affichage	`titre`, `nombre de pages` (non affiché)
les critères sur les lignes	`[nombres de pages] <= 100`
les tris et leur sens	aucun

TAB. 14 – *Requête basée sur une vue (qui est une autre requête déjà enregistrée)*

Comme cette nouvelle requête est mono-table, il n'est pas nécessaire d'effectuer de liaison entre la vue `livres_empruntables` et une autre table. Par contre, si une requête est basée sur plusieurs tables parmi lesquelles se trouve une vue, alors le schéma relationnel de la base ne sait pas comment cette vue est liée aux autres tables. Elle se retrouve alors souvent isolée sur le schéma relationnel de la requête en cours de création (figure 38), alors que, rappelons-le, aucun élément ne doit être isolé sur le schéma relationnel d'une requête.

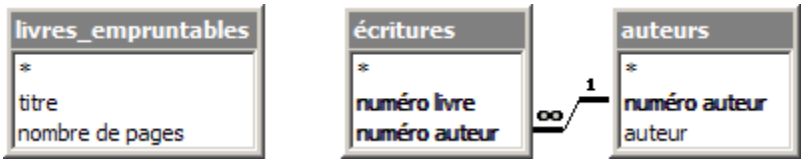
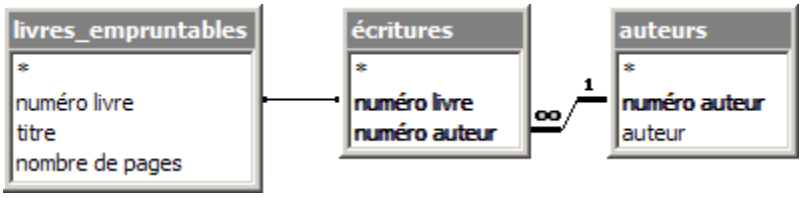


FIG. 38 – *Vue isolée sur le schéma d'une nouvelle requête*

Pour afficher les auteurs des livres empruntables, c'est au développeur de la requête de préciser comment lier la table des écritures à `livres_empruntables`. Or pour cela, il lui manque la colonne `numéro livre`, il doit donc préalablement l'ajouter à la requête `livres_empruntables` avant de poursuivre la nouvelle requête.

Une fois la requête du tableau 11 (page 31) modifiée, de retour dans la nouvelle requête (tableau 15), le programmeur peut alors artificiellement tirer[17] la liaison voulue à l'aide du curseur illisible .

les tables et leurs liaisons	
les colonnes et leur affichage	`titre`, `auteur`
les critères sur les lignes	aucun
les tris et leur sens	aucun

TAB. 15 – *Liaison artificielle sur le schéma d'une requête*

17. en suivant la même procédure que dans l'outil `Relations...`

Il s'agit-là d'une règle générale : une vue sans clé est systématiquement isolée sur le schéma relationnel d'une requête et pour pouvoir être liée aux autres tables, cette vue doit comporter toutes les clés nécessaires, notamment une clé primaire.

3.4.2 Jointure

Nous appelons jointure les liaisons, artificielles ou non, qui figurent sur le schéma relationnel d'une requête, entre les tables et les vues impliquées par la requête. Comme nous venons de le voir pour les vue, une jointure n'est pas forcément une liaison existante sur le schéma relationnel de la base.

Cependant, Access se fonde sur le schéma relationnel de la base pour proposer automatiquement des jointures entre les tables impliquées dans la requête. L'élaboration de ce schéma global est donc important, car il automatise une partie du travail d'élaboration des schémas relationnels des requêtes.

En revanche, le développeur d'une requête ne doit pas être totalement inféodé au schéma relationnel de la base et doit être capable d'améliorer, le cas échéant, le schéma de sa requête.

À titre d'illustration, si nous voulons afficher la liste des parieurs qui ont misé sur le cheval `Petite Julia` dans la course n°4 (figure 39), alors les tables `parieurs`, `paris` et `chevaux` suffisent.

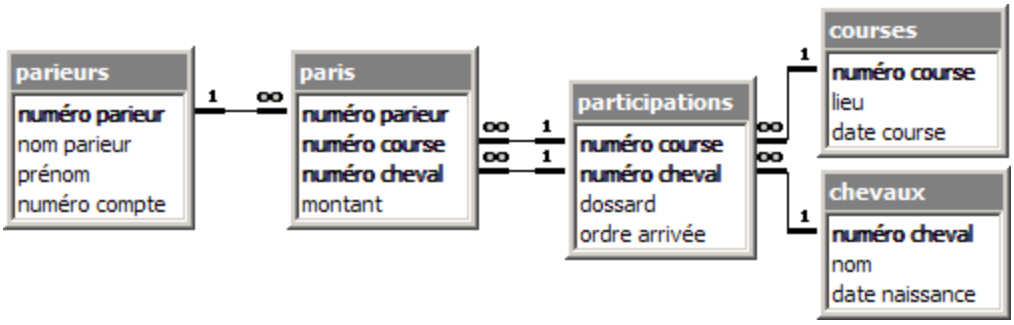
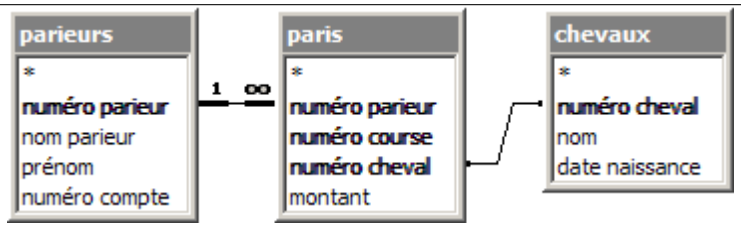


FIG. 39 – *Rappel du schéma relationnel concernant les paris de courses de chevaux*

Nous pouvons donc faire l'économie de la table `participations`, ce qui oblige le développeur de la requête à lier artificiellement[18] la table `chevaux` directement à la table `paris` (tableau 16).

les tables et leurs liaisons	
les colonnes et leur affichage	`parieurs.*`, `numéro course` (non affichée), `chevaux.nom` (non affichée)
les critères sur les lignes	`[numéro course] = 4`, `chevaux.nom = "Petite Julia"`
les tris et leur sens	aucun

TAB. 16 – *Court-circuit d'une table pour économiser une jointure*

Il s'agit d'un court-circuit relationnel tout à fait légitime pour cette requête. Par contre, si nous ne connaissions que le dossard du cheval, mais pas son nom, alors la présence de cette table `participations` aurait été obligatoire.

18. en réalité, Access est capable de deviner automatiquement cette jointure par similitude des intitulés de colonne

3.4.3 Conclusion

La notion de vue est très importante car elle permet de simplifier la programmation d'une requête complexe en la décomposant en plusieurs requêtes simples[19]. Certaines requêtes deviennent même possibles grâce aux vues alors qu'elles sont impossibles à programmer directement. Un exemple est donné section 3.7.4 (page 43).

Les vues permettent également de traduire les intitulés des colonnes et des tables en plusieurs langues et en plusieurs langages métiers. Mais, étant donné qu'une base Access est rarement utilisée par des utilisateurs de langues différentes et/ou de langages métier différents, cette fonctionnalité est peu utilisée en pratique.

Par contre, la possibilité d'enregistrer à l'avance les requêtes sous un nom, sera intensivement utilisée dans l'élaboration des formulaires, à la fois pour sélectionner les données manipulées par le formulaire et pour remplir ses listes (déroulantes ou non).

3.5 Opérations

Une requête peut faire appel aux opérateurs arithmétiques classiques (`+ - * /`), mais aussi `^` (puissance), `\` (division entière : `5\3 = 1`) et `MOD` (modulo, c'est-à-dire le reste par division entière : `5 MOD 3 = 2`).

Cela permet notamment d'afficher des colonnes calculées à partir des colonnes stockées dans les tables. Par exemple, pour afficher le salaire net approximatif (calculé à partir du salaire brut) de chaque employé, il suffit de choisir un intitulé (`salaire net`) pour cette nouvelle colonne, deux points et la formule (tableau 17).

les tables et leurs liaisons	`employés`
les colonnes et leur affichage	`numéro employé`, `salaire net: [salaire brut] * 0.76`
les critères sur les lignes	aucun
les tris et leur sens	aucun

Tab. 17 – *Affichage d'une colonne calculée*

L'intitulé `salaire net` est ensuite utilisable comme n'importe quel intitulé de colonne, pour effectuer une condition de liaison, un tri, ou lorsque la requête est utilisée en tant que vue dans une autre requête.

D'autres opérations sont possibles, notamment la concaténation[20] de chaînes de caractères avec l'esperluette `&`. Par exemple, pour afficher le prénom et le nom des employés dans une seule colonne (classée par ordre alphabétique des noms) la solution se trouve dans le tableau 18.

les tables et leurs liaisons	`employés`
les colonnes et leur affichage	`identité: [prénom employé] & ' ' & [nom employé]`, `nom employé` (non affiché)
les critères sur les lignes	aucun
les tris et leur sens	`nom employé` (croissant)

Tab. 18 – *Concaténation de deux chaînes de caractères*

19. la notion de sous-requête, quoique généreuse en exercices, est rendue inutile par la notion de vue que nous pouvons systématiquement lui substituer
20. la concaténation est la mise bout-à-bout

Remarques :
- la petite chaîne de caractère `' '` permet de s'assurer qu'il y aura au moins une espace entre chaque prénom et chaque nom ;
- si le tri avait été appliqué à la colonne `identité`, alors le tri aurait d'abord tenu compte des prénoms, ce qui n'était pas demandé.

Une autre famille d'opérateurs peut être utilisée, il s'agit des fonctions d'agrégation, notamment : `Min`, `Max`, `Somme`, `Moyenne` et `Compte` (cette dernière permet d'effectuer un dénombrement). Par exemple, pour afficher le salaire brut moyen des employés, il suffit d'utiliser la fonction `Moyenne` (tableau 19).

les tables et leurs liaisons	`employés`
les colonnes et leur affichage	`salaire brut moyen: Moyenne([salaire brut])`
les critères sur les lignes	aucun
les tris et leur sens	inutile (le résultat de la requête ne comporte qu'une ligne)

Tab. 19 – *Calcul d'une moyenne*

Autre exemple, pour afficher le nombre d'employés, c'est la fonction `Compte` sur la clé primaire qui convient (tableau 20).

les tables et leurs liaisons	`employés`
les colonnes et leur affichage	`nombre d'employés: Compte([numéro employé])`
les critères sur les lignes	aucun
les tris et leur sens	inutile

Tab. 20 – *Dénombrement*

Finalement, grâce à ces opérateurs utilisables dans les requêtes, le mieux est de ne stocker dans les tables que les colonnes essentielles et de ne stocker aucune colonne calculable à partir des autres.

3.6 Critères

Les conditions de sélection, autrement dit, les critères sur les lignes, peuvent utiliser tous les opérateurs précédents ainsi que les opérateurs de comparaisons classiques (`< <= = >= >`), l'opérateur « différent de » (`<>`), les opérateurs logiques (`Et`, `Ou`, `Pas`) et les opérateurs `Entre`, `In` et `Comme`.

Par exemple, pour afficher les livres « acquis entre le premier janvier 2004 et aujourd'hui, dont le nombre de pages n'est pas 234 » ainsi que tous livres « acquis avant 2004 et dont le nombre de pages est 234 » (pourquoi pas ?), les critères sur les lignes sont donnés dans le tableau 21.

les tables et leurs liaisons	`livres`
les colonnes et leur affichage	`*`, `date acquisition` (non affichée), `nombre de pages` (non affiché)
les critères sur les lignes	{`[date acquisition] Entre #01/01/2004# Et Date`, `[nombre de pages] <> 234`} `Ou` {`[date acquisition] < #01/01/2004#`, `[nombre de pages] = 234`}
les tris et leur sens	aucun

Tab. 21 – *Articulation de critères sur deux colonnes*

Remarques :
- les dates sont décrites entre dièses `#` et selon le format européen[21] (jour/mois/année) ;
- la fonction `Date` retourne la date courante (et l'heure courante) de l'ordinateur.

L'équivalent graphique Access de la requête précédente est donné sur la figure 40. Il permet de comprendre l'utilité des nombreuses rangées réservées aux critères.

Champ :	livres.*	date acquisition	nombre de pages
Table :	livres	livres	livres
Tri :			
Afficher :	☑	☐	☐
Critères :		Entre #01/01/2004# Et "Date"	<>234
Ou :		<#01/01/2004#	234

FIG. 40 – *Critères écrits sur plusieurs rangées*

Autre exemple, pour afficher les livres dont la date d'acquisition est soit le 23/08/2003, soit le 12/03/1999, soit le 09/12/2001, nous pouvons utiliser l'opérateur `In`, suivi d'une liste de valeurs séparées par des points-virgules (tableau 22).

les tables et leurs liaisons	`livres`
les colonnes et leur affichage	`*`, `date acquisition` (non affichée)
les critères sur les lignes	`date acquisition In (#23/08/2003#; #12/03/1999#; #09/12/2001#)`
les tris et leur sens	aucun

TAB. 22 – *Critère reposant sur une liste de valeurs recherchées*

Par contre, pour afficher les titres qui contiennent le mot `Soleil`, nous en ignorons la liste. Nous ne pouvons donc pas utiliser l'opérateur `In`. Mais, grâce à l'opérateur `Comme` et au joker `*` qui remplace toute chaîne de caractère(s) (y compris vide), nous pouvons filtrer la colonne `titre` de la table `livres` (tableau 23).

les tables et leurs liaisons	`livres`
les colonnes et leur affichage	`*`, `titre` (non affiché)
les critères sur les lignes	`titre Comme "*Soleil*"`
les tris et leur sens	aucun

TAB. 23 – *Utilisation d'un filtre sur une colonne de type chaîne de caractères*

Dernier exemple, pour sélectionner les auteurs dont le nom commence par `Dupon`, mais dont nous ignorons la dernière lettre, c'est le joker `?`, qui remplace un et un seul caractère, qu'il faut employer (tableau 24).

les tables et leurs liaisons	`auteurs`
les colonnes et leur affichage	`auteur`
les critères sur les lignes	`auteur Comme "Dupon?"`
les tris et leur sens	aucun

TAB. 24 – *Autre joker pour les filtres*

La définition des filtres est sensible à la casse, autrement dit, il faut faire attention aux minuscules et aux majuscules que nous utilisons. Dans l'exemple précédent, `dupont` est exclut du filtre, car la lettre initiale `d` est minuscule, contrairement au filtre.

21. sauf si les paramètres régionaux de Windows en décident autrement

3.7 Groupements

Il est parfois nécessaire de regrouper les lignes d'une table selon une catégorie. Par exemple, pour afficher le nombre d'avions de chaque compagnie, il faut regrouper les lignes de la table `avions` selon la valeur de la colonne `numéro compagnie` et en profiter pour dénombrer les lignes qui possèdent le même `numéro compagnie` (tableau 25).

les tables et leurs liaisons	`avions`
les colonnes et leur affichage	`numéro compagnie`, `nombre d'avions: [numéro avion]`
les critères sur les lignes	aucun
les tris et leur sens	aucun
les opérations	`Regroupement` sur `[numéro compagnie]`, `Compte` sur `[numéro avion]`

TAB. 25 – *Groupement et dénombrement*

Pour faire apparaître la rangée des opérations (figure 41), il faut passer en mode `Totaux` à l'aide du bouton Σ.

Champ :	numéro compagnie	nombre d'avions: numéro avion
Table :	avions	avions
Opération :	Regroupement	Compte
Tri :		
Afficher :	☑	☐
Critères :		

FIG. 41 – *Mode `Totaux` et opérations*

À partir du moment où le mode `Totaux` est activé, toutes les colonnes doivent posséder une opération parmi `Regroupement` et les fonctions d'agrégation (`Min`, `Max`, `Somme`, `Moyenne` et `Compte`). C'est pour cette raison que, dans le tableau 25, l'opération `Compte` figure, non pas dans la description des colonnes comme d'habitude, mais dans la description des opérations.

Côté SQL, le passage en mode `Totaux` correspond à l'ajout, dans la requête `SELECT`, de la clause `GROUP BY` pour y préciser les colonnes de `Regroupement`. Par exemple, la syntaxe SQL de la requête ci-dessus est la suivante :

```
1 SELECT [numéro compagnie], Count([numéro avion]) AS [nombre d'avions]
2 FROM avions
3 GROUP BY [numéro compagnie] ;
```

Le mot-clé `AS` permet de préciser l'intitulé (autrement dit, l'alias) de la colonne calculée.

Le passage en mode `Totaux` induit d'autres changements dans l'écriture de la requête. Ces changements, qui concernent notamment les critères de sélection et les fonctions d'agrégation, sont décrits dans les sections suivantes.

3.7.1 Critères sur les lignes et méthodologie

Pour limiter l'affichage du résultat aux compagnies qui possèdent strictement plus de 10 avions et pour limiter le décompte aux avions de type `A320` (type numéro 1), nous utilisons simplement les critères sur les lignes (tableau 26).

les tables et leurs liaisons	`avions`
les colonnes et leur affichage	`numéro compagnie`, `nombre d'avions: [numéro avion]`, `numéro type avion` (non affichée)
les critères sur les lignes	`[nombre d'avions]` $>$ `10`, `numéro type avion = 1`
les tris et leur sens	`nombre d'avions` (croissant)
les opérations	`Regroupement` sur `[numéro compagnie]`, `Compte` sur `[numéro avion]`, `Où` sur `[numéro type avion]`

TAB. 26 – *Agrégation sur les informations complémentaires d'un groupement*

Cet exemple permet de voir que les critères de sélection après regroupement (`nombre d'avions` $>$ `10`) sont différents des critères avant regroupement (`type` $=$ 1), pour lesquels il est nécessaire d'employer l'opération `Où`. En SQL, nous retrouvons la même distinction puisque les conditions de sélection sont réparties entre les clauses `WHERE` et `HAVING` :

```
1 SELECT [numéro compagnie], COUNT([numéro avion]) AS [nombre d'avions]
2 FROM avions
3 WHERE [numéro type avion] = 1
4 GROUP BY [numéro compagnie]
5 HAVING COUNT([numéro avion]) > 10
6 ORDER BY COUNT([numéro avion]) ASC ;
```

Nous voyons sur cette requête la liste complète des clauses admises par une requête `SELECT`. Il est cependant conseillé de rédiger une requête en réalisant les étapes dans cet ordre :

1. déterminer si, oui ou non, la requête a besoin d'une opération de regroupement ;
2. choisir les tables et, si besoin est, les lier correctement ;
3. choisir ou rédiger les colonnes à afficher, ainsi qu'un intitulé pour les colonnes qui résultent d'un calcul ;
4. rédiger les critères de sélection, en ajoutant éventuellement des colonnes non affichées ;
5. choisir les tris et leur ordre, en ajoutant éventuellement d'autres colonnes non affichées ;
6. choisir, le cas échéant, l'opération appropriée à chaque colonne affichée.

3.7.2 Informations complémentaires

Maintenant, si nous préférons afficher le nom des compagnies aériennes plutôt que leur numéro, la contrainte du mode `Totaux` nous impose d'employer une fonction d'agrégation sur la colonne `nom` de la table `compagnie`. Heureusement, Access offre la fonction d'agrégation `Premier` qui, dans chaque groupe d'avions qui possède le même numéro de compagnie, ne retient que le nom de la compagnie du premier avion du groupe (tableau 27).

les tables et leurs liaisons	`avions`, `compagnies` (ces tables sont correctement liées par le schéma relationnel)
les colonnes et leur affichage	`avions.[numéro compagnie]` (non affichée), `nombre d'avions: avions.[numéro avion]`, `compagnie aérienne: compagnie.nom`
les critères sur les lignes	aucun
les tris et leur sens	aucun
les opérations	`Regroupement` sur `avions.[numéro compagnie]`, `Compte` sur `avions.[numéro avion]`, `Premier` sur `compagnie.nom`

TAB. 27 – *Agrégation sur les informations complémentaires d'un groupement*

Remarque : dans l'exemple précédent, nous aurions pu choisir le dernier avion de chaque groupe (avec la fonction d'agrégation `Dernier`), puisque tous les avions d'un même groupe possède le même nom de compagnie.

3.7.3 Regroupements sur plusieurs colonnes

Par ailleurs, nous pouvons effectuer des regroupements sur plusieurs colonnes. Par exemple, pour afficher le nombre d'avions par compagnie aérienne <u>et</u> par type d'avions, il faut grouper les lignes de la table `avions` par la colonne `compagnie` et effectuer des sous-groupes par la colonne `type` (tableau 28).

les tables et leurs liaisons	`avions`
les colonnes et leur affichage	`numéro compagnie`, `numéro type avion`, `nombre d'avions: [numéro avion]`
les critères sur les lignes	aucun
les tris et leur sens	aucun
les opérations	`Regroupement` sur `[numéro compagnie]`, `Regroupement` sur `numéro type avion`, `Compte` sur `numéro avion`

TAB. 28 – *Groupement sur plusieurs colonnes*

Le fait de regrouper les informations sur deux, trois colonnes ou plus est le signe que la requête possède un rôle analytique. Dans ce cas, il est préférable d'effectuer l'analyse à travers un tableau croisé dynamique ou un graphique croisé dynamique, pour lesquels nous renvoyons à nouveau le lecteur vers la référence [Isoz].

3.7.4 Vues préalables

Certaines requêtes requiert parfois la construction d'une vue préalable, sans quoi elles sont impossibles à réaliser. Par exemple, si nous voulons afficher, pour chaque type d'avion, le numéro de la compagnie qui en possède le plus, il s'agit d'une requête très complexe. Nous pouvons la simplifier en utilisant la requête précédente (tableau 28, requête nommée `NombreAvionsParCompagnieEtParType`), dans une nouvelle requête (nommée `MaxAvionsParType`) qui calcule les quantités maximales (tableau 29).

les tables et leurs liaisons	`NombreAvionsParCompagnieEtParType`
les colonnes et leur affichage	`numéro type avion`, `maximum:[nombre d'avions]`
les critères sur les lignes	aucun
les tris et leur sens	aucun
les opérations	`Regroupement` sur `numéro type avion`, `Max` sur `[nombre d'avions]`

TAB. 29 – *Utilisation d'une vue préalable pour calculer des maxima*

Grâce à ces maxima utilisables dans un critère sur les lignes, nous pouvons retirer aux lignes du résultat de la première requête `NombreAvionsParCompagnieEtParType`, les compagnies qui ne réalisent pas le maximum. Il suffit d'écrire la troisième requête (tableau 30) qui elle celle qui répond à notre besoin de départ.

les tables	NombreAvionsParCompagnieEtParType: * ; numéro compagnie ; numéro type avion ; nombre d'avions — MaxAvionsParType: * ; numéro type avion ; maximum
les colonnes et leur affichage	`MaxAvionsParType.[numéro type avion]`, `MaxAvionsParType.maximum`, `NombreAvionsParCompagnieEtParType.[numéro compagnie]`, `NombreAvionsParCompagnieEtParType.[nombre d'avions]` (non affichée)
les critères sur les lignes	`NombreAvionsParCompagnieEtParType.[nombre d'avions] = MaxAvionsParType.maximum`
les tris et leur sens	`MaxAvionsParType.[numéro type avion]` (croissant), `NombreAvionsParCompagnieEtParType.[numéro compagnie]` (croissant)

TAB. 30 – *Utilisation des maxima pour retirer les compagnies qui ne les réalisent pas*

Remarquons que cette dernière requête, qui est une requête simple, sans opération, permet également d'afficher les compagnies ex-aequo.

3.7.5 Conclusion

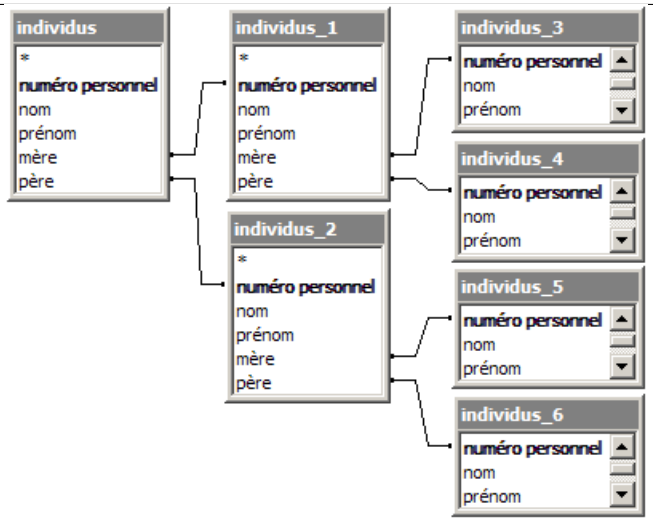
Nous venons de consacrer beaucoup d'effort à la rédaction des requêtes `GROUP BY`. Ces requêtes de sélection, avec groupement(s) et opération(s) d'agrégation, ont un rôle essentiellement analytique vis-à-vis des données, car elles permettent d'en extraire des informations globales, tandis que des requêtes destinées à la saisie ou à la manipulation des données, interviennent au niveau le plus bas. Mais il arrive que des formulaires de saisie aient besoin d'afficher, à l'utilisateur, des informations globales qui vont influencer sa saisie. Par exemple, si le montant total d'une commande dépasse son budget, un acheteur peut vouloir retirer des articles. Dans ce type de configuration, les informations globales (comme le montant total) sont calculées par des agrégations, ce qui justifie l'emploi des requêtes `GROUP BY` en dehors du cadre analytique.

3.8 Auto-jointure

Dans certaines requêtes, une même table peut avoir plusieurs rôles à jouer, auquel cas elle apparaît en plusieurs exemplaires dans le schéma de la requête. C'est souvent le cas lorsque cette table est déjà l'objet d'une auto-liaison sur le schéma relationnel de la base (nous reprenons d'ailleurs l'exemple de la section 2.8 page 26).

À partir du moment où une table est dupliquée sur le schéma d'une requête, les duplicata doivent être liés manuellement au reste du schéma, car Access peut les laisser isolés. Ces liaisons sont donc systématiquement artificielles et une liaison entre deux exemplaires d'une même table est appelée auto-jointure.

Par exemple, affichons les prénoms des parents et des grands-parents de chaque individu (figure 28 page 26), dont nous affichons également le nom de famille et le prénom. Pour cela, la table `individus` joue 7 rôles différents puisqu'elle contient à la fois : la liste des individus, celle des mères, celle des pères, celles des grands-mères maternelles et paternelles puis des grands-pères maternels et paternels (tableau 31).

les tables et leurs liaisons	
les colonnes et leur affichage	`individus.nom`, `individus.prénom`, `mère: individus_1.prénom`, `père: individus_2.prénom`, `grand-mère maternelle: individus_3.prénom`, `grand-père maternel: individus_4.prénom`, `grand-mère paternelle: individus_5.prénom`, `grand-père paternel: individus_5.prénom`
les critères sur les lignes	aucun
les tris et leur sens	aucun

TAB. 31 – *Auto-jointures issues d'une auto-liaison*

Toutes les liaisons qui figurent sur le schéma relationnel de cette requête sont artificielles et doivent donc être organisées par le développeur de la requête. Pour cela, le `Mode création` d'Access est un outil visuel qui permet de réaliser cette opération de manière très ergonomique, contrairement à une rédaction directe en langage SQL.

Malheureusement, ces liaisons artificielles sont, au même titre que toutes les jointures que nous avons rencontrées jusqu'ici, considérées par Access comme étant obligatoires. Autrement dit, le résultat de l'exemple précédent élimine les individus pour lesquels les colonnes `mère` ou `père` sont vides[22]. La seule façon d'afficher tous les individus, y compris ceux à qui il manque (dans la table `individus`) un parent ou un grand-parent, est de rendre facultatives les jointures (section suivante).

22. et parmi les individus dont les colonnes `mère` et `père` ne sont pas vides, sont encore éliminés ceux à qui il manque un grand-parent

3.9 Jointures externes

Reprenons le schéma relationnel de la figure 42, où nous avions séparé les informations complémentaires (placées dans la table `coordonnées`) des informations principales (laissées dans la table `individus`).

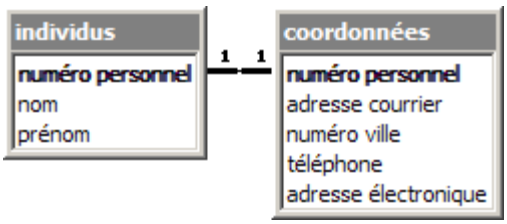


FIG. 42 – *Rappel de l'utilisation de l'héritage pour séparer les colonnes principales des colonnes complémentaires*

Pour afficher la liste des individus, munis de leurs coordonnées (numéro de téléphone et adresse électronique), la requête du tableau 32 ne convient pas, car la jointure entre les deux tables est obligatoire : seuls les individus qui possèdent une ligne dans la table `coordonnées` sont affichés.

les tables et leurs liaisons	`individus`, `coordonnées` (ces tables sont déjà liées par le schéma relationnel)
les colonnes et leur affichage	`nom`, `prénom`, `téléphone`, `adresse électronique`
les critères sur les lignes	aucun
les tris et leur sens	aucun

TAB. 32 – *Jointure interne inappropriée pour afficher des informations complémentaires*

Afin de rendre facultative la jointure avec la table `coordonnées`, il faut double-cliquer sur la liaison entre les deux tables afin de faire apparaître la boîte de dialogue de la figure 43 qui permet de basculer le type 1 de jointure (valeur par défaut) en type 2 ou 3 (il suffit de bien lire leurs intitulés pour faire la différence entre les trois types).

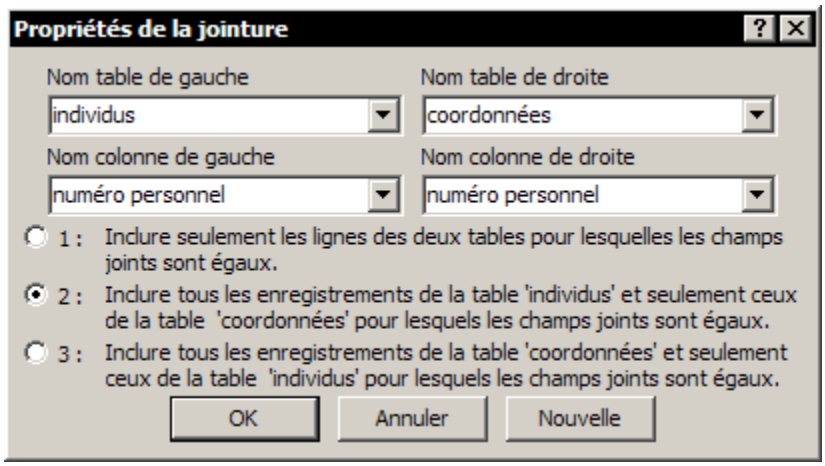


FIG. 43 – *Choix de la bonne jointure externe*

Cette fois, les individus pour lesquels la table `coordonnées` ne contient pas d'informations sont présents dans le résultat de la requête les colonnes `téléphone` et `adresse électronique` sont simplement vides pour eux. Access symbolise une jointure externe par une terminaison en forme de flèche du côté de la table facultative (tableau 33).

les tables	individus: * ; **numéro personnel** ; nom ; prénom — 1 → 1 — coordonnées: * ; **numéro personnel** ; adresse courrier ; numéro ville ; téléphone ; adresse électronique
les colonnes et leur affichage	`nom`, `prénom`, `numéro de téléphone: Nz(coordonnées.téléphone;"inconnu")`, `adresse email: Nz(coordonnées.[adresse électronique];"inconnue")`
les critères	aucun
les tris	aucun

TAB. 33 – *Jointure externe appropriée pour afficher des informations complémentaires*

Pour éviter les valeurs vides à l'affichage du résultat, les colonnes `téléphone` et `adresse électronique` sont légèrement transformées dans cette requête. Il a donc fallu donner un nouvel intitulé à chacune de ces deux colonnes calculées et cet intitulé ne peut pas être identique au nom de départ de la colonne.

3.9.1 Remplacement des valeurs vides

Pour remplacer les valeurs vides des colonnes `téléphone` et `adresse électronique` dans le résultat de la requête, nous avons choisi, dans le tableau 33, d'utiliser la fonction `Nz` qui change à la volée la valeur vide par la valeur qui lui est passée en second argument (la chaîne de caractère `"inconnu"` en l'occurrence).

Il existe de nombreuses fonctions utilisables une requête Access. Pour en parcourir la liste, il suffit d'afficher le générateur d'expression à l'aide du bouton . Ce générateur d'expression offre d'ailleurs davantage d'espace pour saisir la formule qui définit une colonne ou un critère de sélection sur les lignes (figure 44).

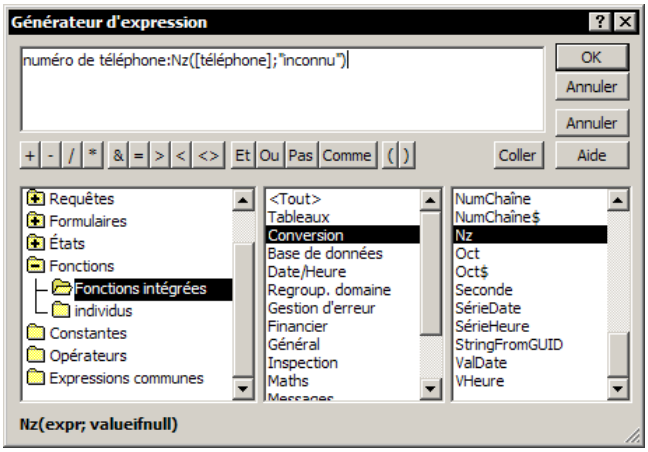


FIG. 44 – *Utilisation du générateur d'expression pour saisir la formule d'une colonne ou d'un critère, à partir des fonctions Access*

Cette ergonomie, bien que toute relative, fait du générateur d'expression un outil conseillé lorsque la formule à saisir est longue et donc source d'erreurs, surtout si elle fait appel à des fonctions intégrées.

3.9.2 Jointure externe plusieurs-à-plusieurs

Pour afficher les livres avec leur(s) catégorie(s) éventuellement (liaison plusieurs-à-plusieurs), nous avons besoin d'effectuer la jointure externe de la figure 45. Malheureusement, Access ne permet pas de joindre une deuxième table du côté de la table facultative. Un message d'erreur apparaît à l'exécution de la requête, parlant de « jointures externes ambiguës ».

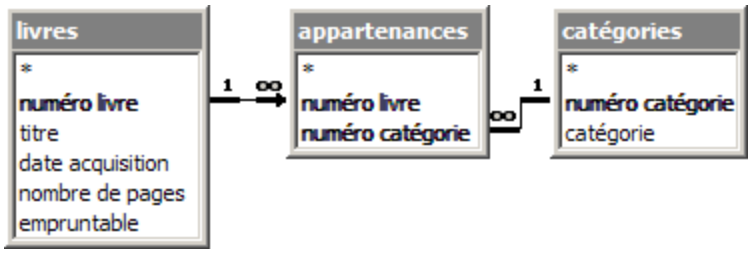


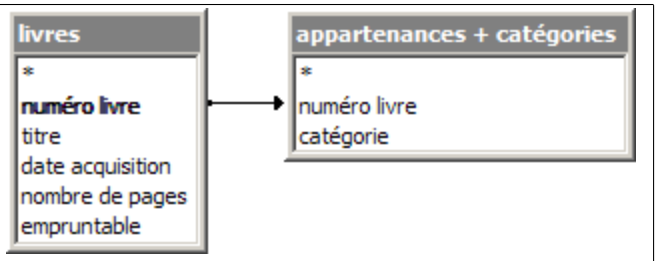
FIG. 45 – *Jointure externe avec une table additionnelle côté facultatif*

La solution consiste simplement à créer une requête préalable qui regroupent toutes les tables du côté facultatif (tableau 34).

les tables et leurs liaisons	`appartenances`, `catégories` (ces tables sont correctement liées par le schéma relationnel)
les colonnes et leur affichage	`numéro livre`, `catégorie`
les critères sur les lignes	aucun
les tris et leur sens	aucun

TAB. 34 – *Vue préalable nommée* `appartenances + catégories` *pour toutes les tables côté facultatif de la jointure externe*

Grâce à cette vue, la jointure externe ne concerne plus qu'une seule table côté facultatif et Access s'y retrouve (tableau 35).

les tables et leurs liaisons	
les colonnes et leur affichage	`titre`, `catégorie`
les critères sur les lignes	aucun
les tris et leur sens	aucun

TAB. 35 – *Jointure externe pour une liaison plusieurs-à-plusieurs*

Cette technique de vue préalable n'est pas obligatoire dans les autres situations, mais est tout de même conseillée, afin de réduire la complexité de certaines requête.

3.9.3 Auto-jointures externes

Revenons à l'exemple de la section 3.8 page 44 et rappelons que le schéma relationnel du tableau 31 ne convient pas totalement, car les individus à qui il manque un parent ou un grand-parent sont éliminés du résultat. La solution pour conserver ces individus consiste à rendre facultatives toutes les jointures (figure 46) sur le schéma relationnel de la requête.

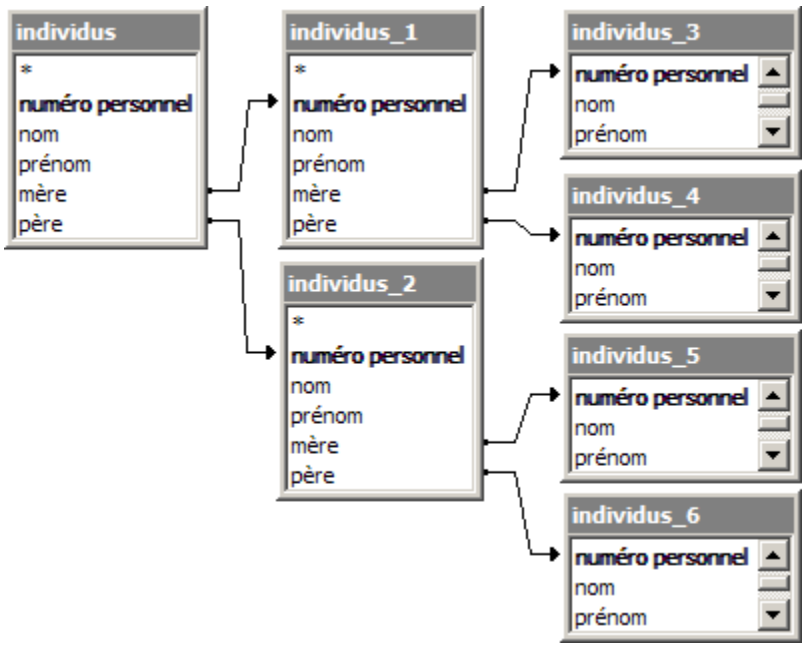


FIG. 46 – *Utilisation d'auto-jointures externes afin de préserver toutes les lignes du résultat*

À nouveau, il est possible de remplacer les éventuelles valeurs vides du résultat à l'aide de la fonction de dénullification `Nz`.

3.9.4 Élimination des cycles

Considérons une nouvelle base dont le schéma relationnel est celui de la figure 47. Nous voulons afficher la liste des formateurs qui animent la formation n°123, ainsi que leur rémunération (en tenant compte d'une éventuelle prime d'organisation).

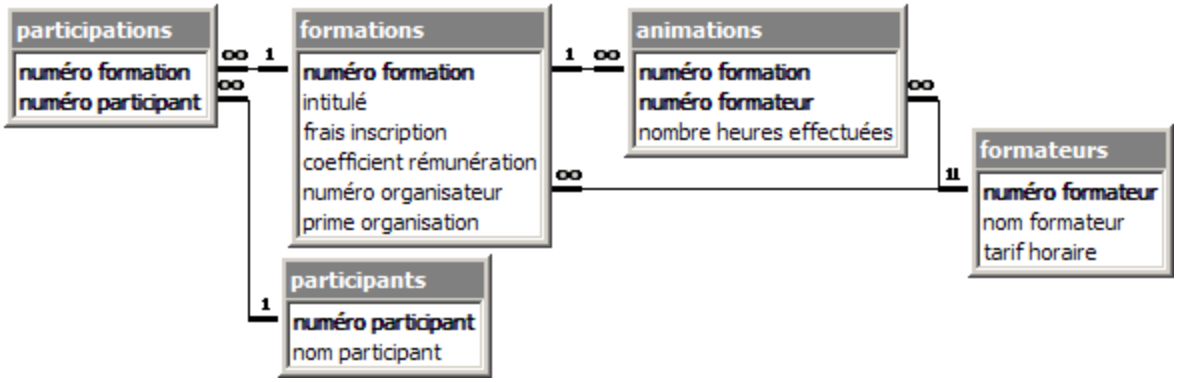
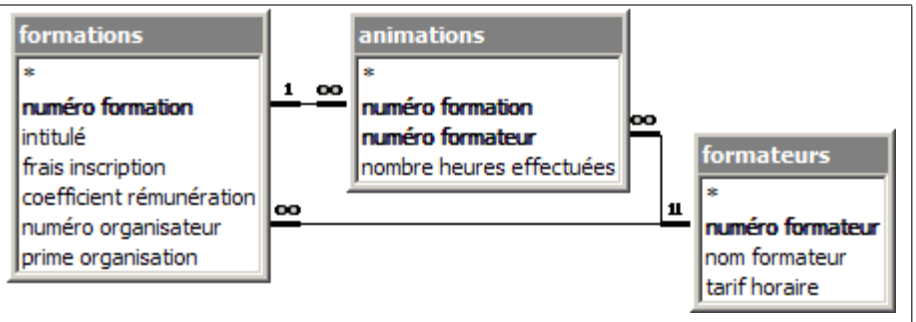


FIG. 47 – *Présence d'un cycle sur le schéma relationnel global d'une base de données*

La requête du tableau 36 ne convient pas, car le mécanisme des jointures en forme de cycle entre les trois tables, élimine du résultat les formateurs qui ne sont pas organisateurs. En effet, la liaison directe entre les tables `formations` et `formateurs` est nécessaire, car elle permet de prendre en compte la `prime d'organisation` dans le calcul de la `rémunération`. Mais comme, par défaut, cette jointure est obligatoire, les formateurs qui ne sont pas organisateurs sont exclus du résultat.

les tables et leurs liaisons	

les colonnes et leur affichage	`nom formateur`, `numéro formation` (non affichée),
	`rémunération: [coefficient rémunération]`
	`* [nombre heures effectuées] * [tarif horaire]`
	`+ [prime organisation]`

les critères sur les lignes	`[numéro formation] = 123`
les tris et leur sens	aucun

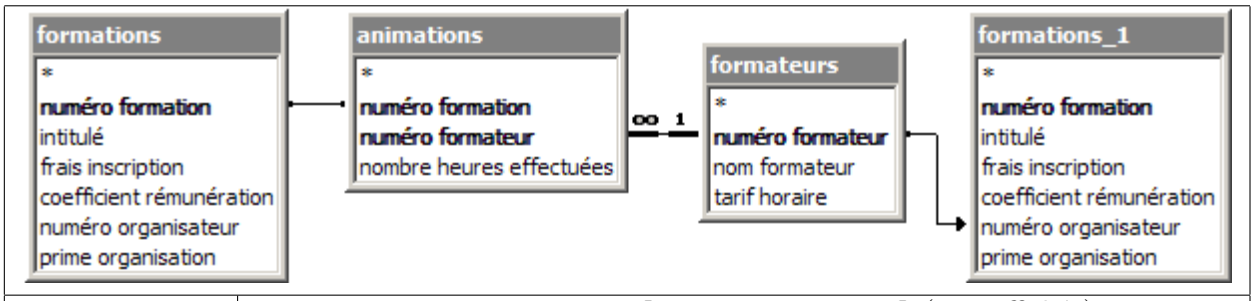
TAB. 36 – *Cycle de jointures problématique*

Le lecteur est invité à tester ce type de requête comportant un cycle pour s'apercevoir que le résultat est partiel. Pour obtenir un résultat complet, il faut donc corriger le schéma relationnel de la requête manuellement.

La solution consiste à mettre à plat le cycle de jointure :

- en ajoutant un deuxième exemplaire de la table `formations` (bouton) ;
- en liant cette table `formations_1` à la table `formateurs` *via* la clé étrangère `numéro organisateur` ;
- en supprimant la liaison directe entre les tables `formations` et `formateurs`.

Puis il faut rendre facultative la jointure entre les tables `formations_1` et `formateurs` afin qu'un formateur non organisateur puisse apparaître dans le résultat (tableau 37).



les colonnes et leur affichage	`nom formateur`, `formations.[numéro formation]` (non affichée), `rémunération: formations.[coefficient rémunération] * [nombre heures effectuées] * [tarif horaire] + Nz(formations_1.[prime organisation];0)`
les critères sur les lignes	`formations.[numéro formation] = 123`
les tris et leur sens	aucun

TAB. 37 – *Mise à plat du cycle précédent*

L'emploi de la fonction `Nz` pour remplacer l'absence de prime par une prime nulle est indispensable dans cette requête. En effet, toute opération arithmétique avec une valeur `Null` résulte en une valeur vide `Null`, ce qu'il faut bien sûr éviter pour la rémunération des formateurs qui ne sont pas organisateur.

3.10 Conclusion

Nous voyons sur ces derniers exemples, que l'élaboration du schéma relationnel d'une requête de sélection peut se révéler aussi complexe que l'élaboration du schéma relationnel de la base de données lui-même. C'est également le cas pour les requêtes d'action de la section suivante.

Pour le lecteur désireux de se perfectionner en la matière, la lecture de [Brouard et Soutou] sur le langage SQL standard, est vivement conseillée. Pour le développeur Access, la rédaction directe d'une requête en SQL offre, en effet, davantage de possibilités que le `Mode création`. C'est le cas, notamment, des inégalités parfois requises dans les conditions de jointure[23].

23. dans tous nos exemples de requête, les conditions de jointures n'étaient que des égalités, souvent entre une clé étrangère et une clé primaire (cela ne suffit pas toujours)

4 Requêtes d'action

Pour insérer, supprimer ou mettre à jour les données d'une ligne à la fois, nous pouvons saisir ces données manuellement dans un formulaire ou directement dans une table. Il est également possible de le faire pour plusieurs lignes à la fois (mais dans une seule table), à l'aide d'une requête de sélection transformée en requête d'action (figure 48).

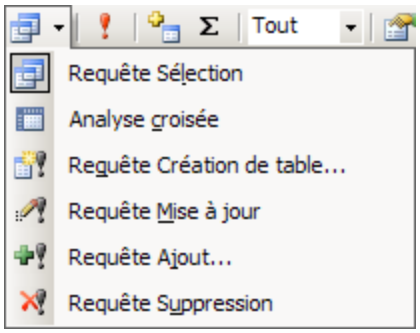


FIG. 48 – *Les différents types de requête*

Les actions possibles sur les tables sont classées en quatre catégories de requêtes :
- l'insertion de nouvelles lignes dans une table, à partir des données déjà présentes par ailleurs ;
- la suppression de certaines lignes déjà présentes dans une table, en fonction de critères spécifiques ;
- la mise à jour de certaines valeurs en effectuant un calcul sur les données ;
- et la création d'une nouvelle table, *via* son remplissage par le résultat d'une requête de sélection.

Une fois enregistrées, ces requêtes d'action peuvent être exécutées :
- à la demande de l'utilisateur (en cliquant sur un bouton dans un formulaire, par exemple) ;
- à la demande du développeur de la requête, en cliquant sur le bouton ;
- ou tout simplement en ouvrant la requête (sauf en mode `Création`).

Le message de confirmation qui apparaît alors, peut être évité en décochant les cases de l'encart `Confirmer` (figure 49), disponible dans le menu `Outils`, `Options...`, onglet `Modifier/Rechercher`.

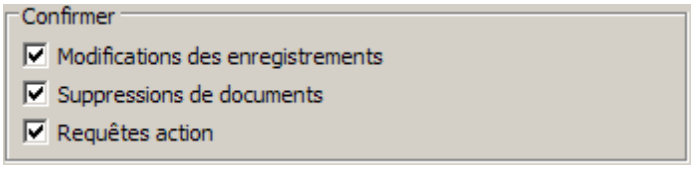


FIG. 49 – *Cases à décocher pour éviter les messages de confirmation à l'exécution d'une requête d'action*

Il ne faut pas oublier de le faire pour tous les utilisateurs Access du poste.

4.1 Insertion (ajout)

Considérons que nous venons de créer la table `livres non empruntables` (figure 50) qui possède les colonnes `numéro livre` (en numérotation automatique), `ancien numéro`, `titre`, `date d'acquisition` et `nombre de pages`. Nous souhaitons insérer dans cette nouvelle table, tous les livres non empruntables de la table `livres`.

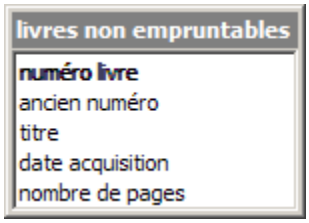


FIG. 50 – *Table destinée à être remplie par une requête d'insertion*

Pour cela, nous créons une nouvelle requête de sélection qui, de la table `livres`, sélectionne les lignes marquées non empruntables et affiche les colonnes `numéro livre`, `titre`, `date d'acquisition` et `nombre de pages`. Puis nous transformons cette requête de sélection en requête d'insertion à l'aide du bouton qui fait apparaître la boîte de dialogue de la figure 51, afin de choisir la table de destination.

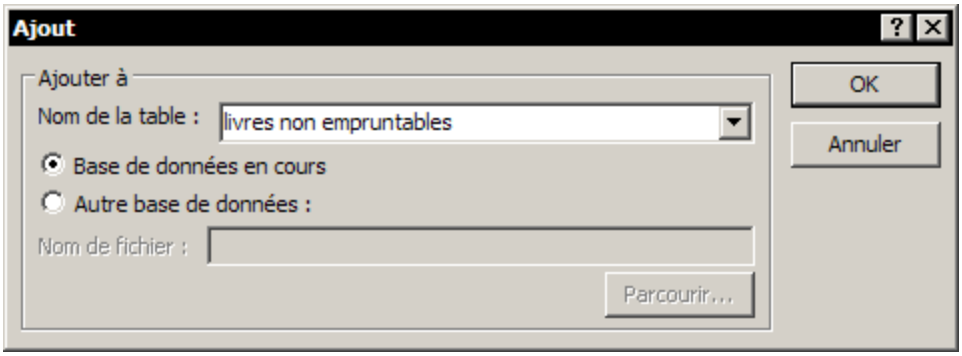


FIG. 51 – *Boîte de dialogue qui apparaît lors de la conversion d'une requête de sélection en une requête d'insertion*

Il suffit alors de préciser dans quelle table l'insertion doit être effectuée (ici, il s'agit de la table `livres non empruntables`). Apparaît ensuite la rangée `Ajouter à`, dans laquelle nous précisons les colonnes de destination (tableau 38).

les tables source	`livres`
la table destination	`livres non empruntables`
les colonnes transférées	`[numéro livre]` → `[ancien numéro]`, `titre` → `titre`, `[date acquisition]` → `[date acquisition]`, `[nombre de pages]` → `[nombre de pages]`
les critères sur les lignes	`empruntable = Oui`
les tris et leur sens	aucun (rarement utile dans ce type de requête)

TAB. 38 – *Requête d'insertion*

Dans une requête d'insertion, il n'est pas possible d'insérer des valeurs dans une colonne en numérotation automatique. C'est le cas de la colonne `numéro livre` de la table `livres non empruntables` qui renumérote les livres fraîchement insérés. Or, nous aurons besoin, dans la suite, de connaître l'ancien `numéro livre`. C'est pour cette raison que nous l'avons transféré dans la colonne additionnelle `ancien numéro`.

Par ailleurs, la requête d'insertion peut faire appel à plusieurs tables (ou requêtes) liées entre elles, à des opérations sur les colonnes, aux critères sur les lignes et aux groupements. D'ailleurs, l'interface graphique de saisie d'une requête d'insertion ressemble beaucoup à celle d'une requête de sélection (figure 52).

Champ :	numéro livre	titre	date acquisition	nombre de pages	empruntable
Table :	livres	livres	livres	livres	livres
Tri :					
Ajouter à :	ancien numéro	titre	date acquisition	nombre de pages	
Critères :					=Oui

FIG. 52 – *Requête d'action de type ajout*

En SQL, une requête d'insertion est écrite avec l'instruction `INSERT INTO` suivie :
- du nom de la table de destination ;
- des colonnes à remplir ;
- puis de la requête `SELECT` qui sélectionne les données source.

À titre d'exemple, la traduction SQL de la requête précédente est la suivante :

```
1 INSERT INTO [livres non empruntables]
2   ([ancien numéro], titre, [date acquisition], [nombre de pages])
3 SELECT [numéro livre], titre, [date acquisition], [nombre de pages]
4 FROM livres
5 WHERE empruntable = Yes ;
```

Les données insérées dans la table de destination doivent impérativement vérifier les contraintes de cette table, à savoir :
- les contraintes `Null interdit` (il faut insérer une valeur dans toutes les colonnes qui sont déclarées non vide) ;
- les contraintes `Valide si` (les données insérées doivent vérifier les règles de validation des colonnes et de la table) ;
- les types de données (il ne faut pas insérer une date dans une colonne numérique, par exemple) ;
- et surtout, les clés étrangères (il ne faut pas que les valeurs insérées pointent vers des références qui n'existent pas).

Par exemple, pour connaître les auteurs des livres de la table `livres non empruntables`, nous créons la table `écritures2` (figure 53), puis nous y insérons les lignes à l'aide de la requête d'insertion du tableau 39.

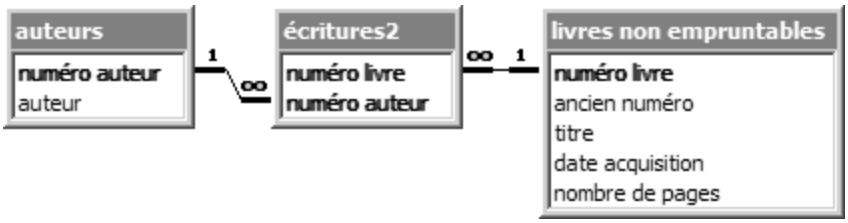
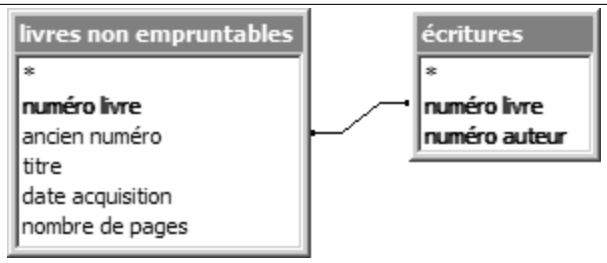


FIG. 53 – *Schéma relationnel pour la table des livres non empruntables*

Dans cette requête, les données source sont contenues dans deux tables non liées par le schéma relationnel de la base. Il est donc nécessaire d'effectuer la liaison artificielle sur le schéma relationnel de la requête, car rappelons-le, aucune table qui figure sur le schéma relationnel d'une requête ne doit être isolée.

les tables source	
la table destination	`écritures2`
les colonnes transférées	`[livres non empruntables].[numéro livre]` → `[numéro livre]`, `écritures.[numéro auteur]` → `[numéro auteur]`
les critères sur les lignes	aucun
les tris et leur sens	aucun

TAB. 39 – *Requête d'insertion dans une table de jonction*

Puisque la table `écritures2` est liée à la table `livres non empruntables` par la clé étrangère `numéro livre`, le `numéro livre` à insérer est effectivement le nouveau numéro automatique qui a été attribué aux livres insérés dans la table `livres non empruntables` et non l'`ancien numéro` qui provenait de la table `livres`. Par contre, cet `ancien numéro` est indispensable pour faire la liaison artificielle entre les livres nouvellement insérés et la table `écritures` originelle.

Le lecteur est invité à bien réfléchir sur cet exemple qui montre comment il est possible d'établir automatiquement de nouvelles liaisons clé étrangère - clé primaire, à partir d'anciennes liaisons clé étrangère - clé primaire, sans créer de clé étrangère orpheline.

Si dans une requête d'insertion le développeur est amené à changer de table de destination, il peut le faire en éditant les propriétés de la requête (comme dans la section 3.2 page 32). Apparaît alors la fenêtre de la figure 54, dans laquelle la rangée `Table destination` permet de choisir la table voulue.

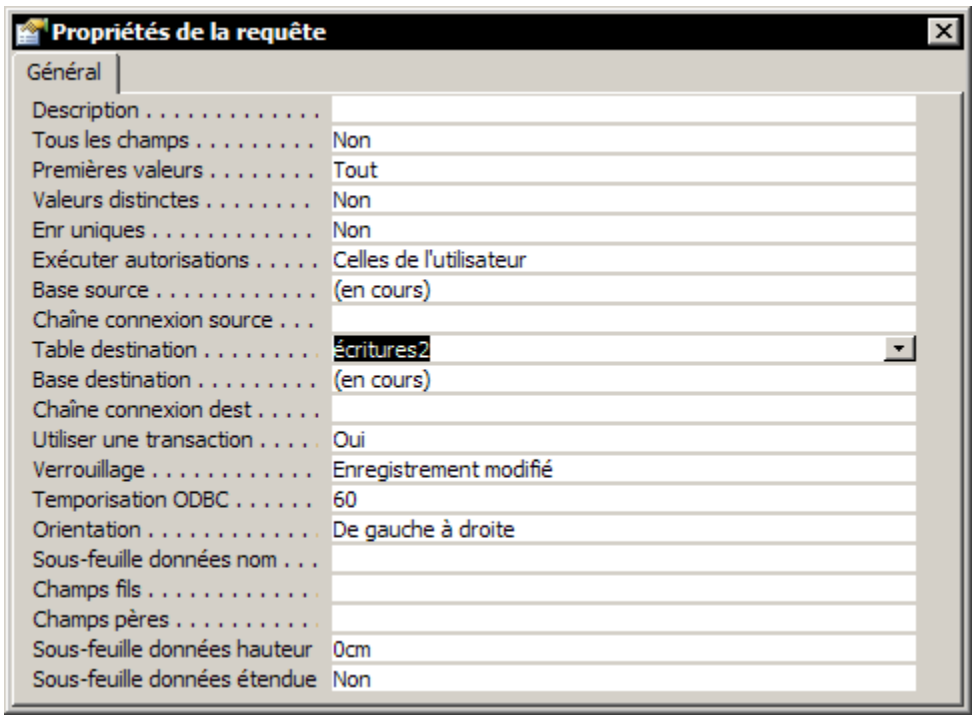


FIG. 54 – *Propriétés d'une requête d'insertion*

Cette table de destination n'a pas besoin d'être liée aux autres tables de la requête (les tables source). C'est pour cette raison que nous réservons une rangée spécifique à cette table de destination dans les tableaux 39 et 38. Mais notons qu'il n'est pas exclut qu'une table de destination serve aussi de table source dans la même requête d'insertion.

4.2 Suppression

Grâce aux insertions de la section précédente, nous pouvons désormais supprimer les livres non empruntables de la tables `livres`. Pour cela, nous créons une nouvelle requête, qui porte sur la table `livres` et sur toutes ses colonnes (`livres.*`). Puis nous transformons cette requête de sélection en requête de suppression à l'aide du bouton . Il suffit alors de préciser les critères de sélection des lignes à supprimer (tableau 40).

les tables et leurs rôles	`livres` (dans laquelle supprimer)
les colonnes	`livres.*` (`A partir de`), `empruntable` (`Où`)
les critères sur les lignes	`empruntable = Oui`

TAB. 40 – *Requête de suppression*

Dans une requête de suppression, nous ne pouvons utiliser ni tri, ni le mode `Totaux`. Par ailleurs, la rangée `Supprimer` doit indiquer `A partir de` pour les colonnes à supprimer (il s'agit toujours de `*`) et `Où` pour les colonnes qui servent de critère de sélection (figure 55).

Champ :	livres.*	empruntable
Table :	livres	livres
Supprimer :	A partir de	Où
Critères :		=Oui

FIG. 55 – *Requête d'action de type suppression*

En SQL, une requête de suppression est écrite avec l'instruction `DELETE` qui ne possède que trois clauses :

- la clause `DELETE` pour préciser les colonnes à supprimer (`*`) et les colonnes de critère de sélection ;
- la clause `FROM` pour préciser la table dans laquelle supprimer, ainsi que les tables qui servent aux critères de sélection ;
- la clause `WHERE` pour préciser les critères de sélection.

Dans notre exemple, la traduction SQL de la requête est la suivante :

```
1 DELETE livres.*, empruntable
2 FROM livres
3 WHERE empruntable = Yes ;
```

Par ailleurs, grâce à l'intégrité référentielle, nous ne pouvons pas supprimer des lignes qui sont référencées par une clé étrangère (sinon, nous aurions des références orphelines).

Dans notre exemple, les livres non empruntables sont encore référencés par la tables `écritures`. Avant d'exécuter la requête précédente, il faut donc exécuter au préalable la requête du tableau 41.

les tables et leurs rôles	`écritures` (dans laquelle supprimer), `livres` (pour les critères) (ces tables sont correctement liées par le schéma relationnel)
les colonnes	`écritures.*` (`A partir de`), `livres.empruntable` (`Où`)
les critères sur les lignes	`livres.empruntable = Oui`

TAB. 41 – *Requête de suppression préalable pour éviter les clés étrangères orphelines*

Il faut être vigilant avec une requête de suppression, car si aucun critère de sélection n'est saisi, alors à son exécution, toutes les lignes sont supprimées.

4.3 Mise à jour

Hormis l'insertion et la suppression, les requêtes d'action permettent également de modifier les données de certaines colonnes, de plusieurs lignes à la fois, mais toujours dans une seule table.

Par exemple, pour augmenter de 10 % le salaire brut des employés qui ont pour chef, l'employé n°8, nous créons une nouvelle requête, basée sur la table `employés` (figure 27 page 26) et sur les colonnes `salaire brut` et `numéro supérieur hiérarchique`. Puis nous transformons cette requête de sélection en requête de mise à jour à l'aide du bouton . Il suffit alors de préciser les critères de sélection des lignes à mettre à jour, ainsi que les formules qui donnent les nouvelles valeurs (tableau 42).

les tables et leurs rôles	`employés` (à mettre à jour)
les mises à jour	`[salaire brut]` ← `[salaire brut] * 1.1`
les critères sur les lignes	`[numéro supérieur hiérarchique] = 8`

TAB. 42 – *Requête de mise à jour*

Un requête d'action de type mise à jour se reconnaît à la rangée `Mise à jour` qui apparaît dans la fenêtre de saisie (figure 56).

Champ :	salaire brut	numéro supérieur hiérarchique
Table :	employés	employés
Mise à jour :	[salaire brut]*1.1	
Critères :		8

FIG. 56 – *Requête d'action de type mise à jour*

En SQL, une requête de mise à jour est écrite avec l'instruction `UPDATE` qui ne possède que trois clauses :

- la clause `UPDATE` pour préciser la table dans laquelle la mise à jour s'effectue et les autres tables éventuelles qui servent aux nouvelles valeurs et aux critères de sélection ;
- la clause `SET` pour préciser les formules qui donnent les nouvelles valeurs ;
- la clause `WHERE` pour préciser les critères de sélection sur les lignes concernées.

La traduction SQL de la requête précédente est la suivante :

```
1 UPDATE employés
2 SET [salaire brut] = [salaire brut] * 1.1
3 WHERE [numéro supérieur hiérarchique] = 8 ;
```

La section suivante propose une requête de mise à jour qui repose sur plusieurs tables. Mais auparavant, il reste un certain nombre de remarques et de recommandations à formuler au sujet de ce type de requête.

Comme pour une requête d'insertion, il est impératif que les nouvelles valeurs vérifient les contraintes de la table dans laquelle elles sont modifiées :

- les contraintes `Null interdit` (il ne faut pas vider une cellule dans une colonne `Null interdit`) ;
- les contraintes `Valide si` (les nouvelles données doivent vérifier les règles de validation de la table et des colonnes modifiées) ;
- les types de données (il est impossible de remplacer une valeur numérique par une date, notamment) ;
- et surtout, les clés étrangères (d'une part, il ne faut pas changer la valeur d'une clé étrangère vers une référence qui n'existe pas, d'autre part, il ne faut pas changer la valeur d'une clé primaire référencée par une clé étrangère sans quoi cette dernière deviendrait orpheline).

Il est donc quasiment impossible (et de toute façon peu recommandé) de modifier la valeur d'une clé primaire. À moins que toutes les clés étrangères qui la référencent ne soit déclarées avec « mise à jour en cascade » (il s'agit d'une option que nous pouvons cocher lorsque nous définissons la liaison clé primaire-clé étrangère sur le schéma relationnel, figure 57). Dans ce cas, la modification de la valeur d'une clé primaire entraîne automatiquement la modification identique de la clé étrangère dont la mise à jour a été déclarée en cascade.

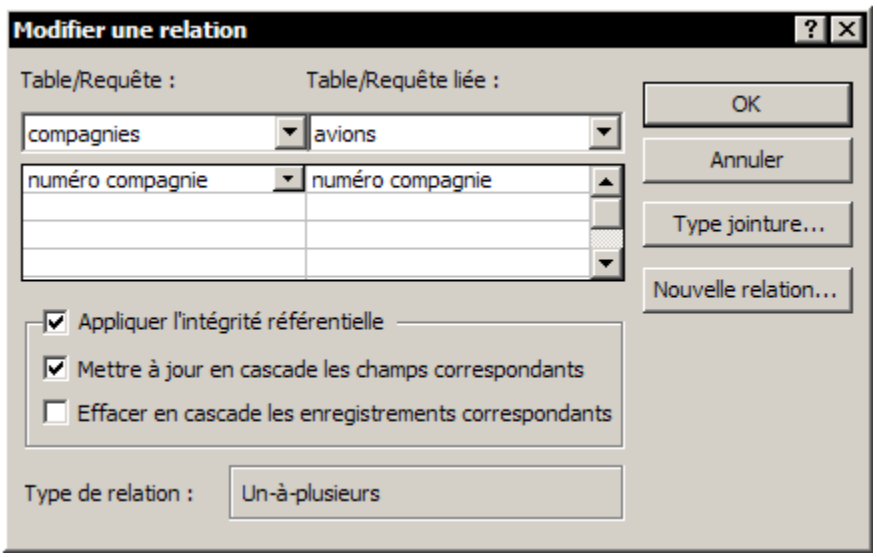


FIG. 57 – *Rappel des réglages conseillés pour l'intégrité référentielle (cette fenêtre est obtenue en double cliquant sur la liaison voulue dans la fenêtre du schéma relationnel)*

Il est donc possible de cocher systématiquement la case « mise à jour en cascade », lorsque nous définissons une liaison clé primaire-clé étrangère sur le schéma relationnel. Cependant, il est vivement déconseillé de cocher la case « suppression en cascade » car cela entraînerait la suppression automatique de la clé étrangère, lorsque la clé primaire est supprimée (éventuellement par erreur).

Revenons aux requêtes de mise à jour, pour constater que nous ne pouvons utiliser ni tri, ni le mode `Totaux`. Si nous avons besoin d'effectuer un regroupement pour calculer les valeurs de mise à jour ou les critères de sélection, alors il faut le faire dans une requête de création de table préalable (section suivante).

4.4 Création de table

La section 3.4 page 35, sur les vues, montre comment une requête de sélection peut être utilisée comme une table, sans qu'aucune table contenant les données qui résultent de la requête ne soient créée. Il est cependant possible de transformer une requête de sélection en une requête de création de table à l'aide du bouton , en précisant le nom de la table à créer. La requête de sélection n'est pas modifiée, mais ses futures exécutions résulteront, non pas en l'affichage du résultat, mais en la (re-)création de la table voulue.

Par exemple, nous ajoutons à la table des auteurs, la colonne `nombre livres` (figure 58) et nous souhaitons la remplir en comptant les livres (empruntables) par auteurs. Il s'agit d'une colonne calculée, dont nous avons déjà dit le plus grand mal (redondance et risque d'incohérence), mais qui constitue la meilleure illustration pour cette section.

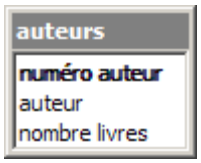


FIG. 58 – *Nouvelle colonne qu'il faudra remplir à l'aide d'un regroupement*

Pour cela, il faut à la fois faire un regroupement sur la table `écritures` et une mise à jour dans la table `auteurs`. Nous sommes donc obligé d'en passer par une requête intermédiaire qui effectue le regroupement seul (tableau 43).

les tables source	`écritures`
le nom de la table à créer	`nombre livres par auteur`
les colonnes et leur affichage	`numéro auteur`, `nombre livres: [numéro livre]`
les critères sur les lignes	aucun
les tris et leur sens	aucun
les opérations	`Regroupement` sur `[numéro auteur]`, `Compte` sur `[numéro livre]`

TAB. 43 – *Requête de création de la table* `nombre livres par auteur`

Cette requête est alors transformée en requête de création de la nouvelle table[24] `nombre livres par auteur` (figure 59).

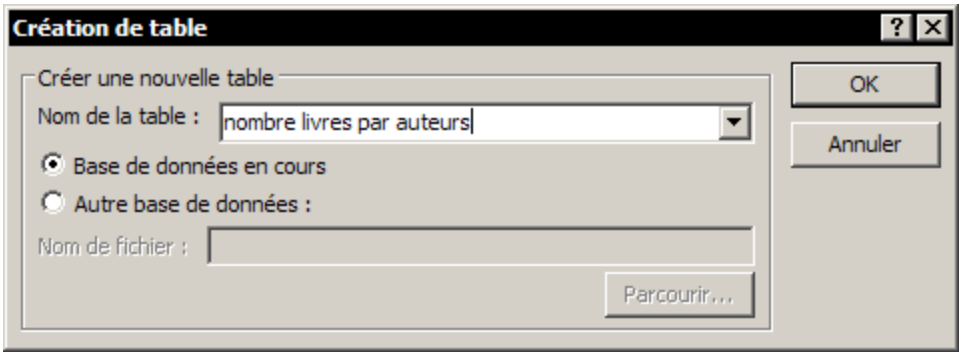


FIG. 59 – *Boîte de dialogue qui apparaît lors de la conversion d'une requête de sélection en une requête de création de table*

24. attention, la requête ne doit pas porter le même nom

Si le développeur d'une requête de création de table a besoin de changer le nom de la table à créer, alors comme précédemment, il peut le faire dans la fenêtre des propriétés de la requête (figure 60).

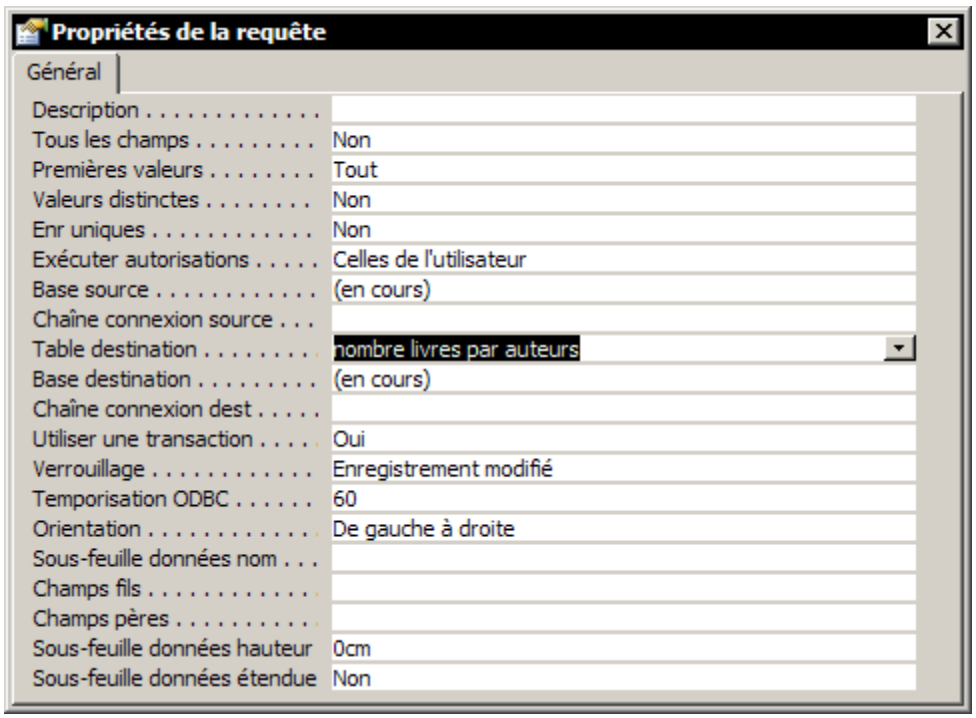
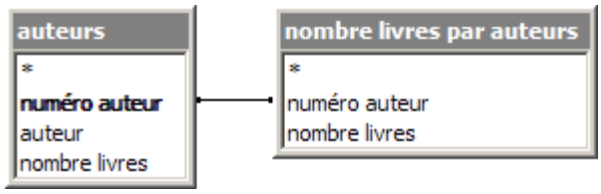


FIG. 60 – *Propriétés d'une requête de création de table*

Revenons à notre exemple, pour constater que nous pouvons ensuite exécuter cette requête de création de table et établir enfin une requête de mise à jour qui utilise la nouvelle table créée (tableau 44).

les tables et leurs rôles	`auteurs` (à mettre à jour), `nombre livres par auteur` (source), liées ainsi : 
les mises à jour	`[nombre livres]` ← `[nombre livres par auteur].[nombre livres]`
les critères sur les lignes	aucun

TAB. 44 – *Formule de mise à jour impliquant une autre table*

Notons au passage que cette requête est notre premier exemple de requête de mise à jour à faire intervenir une autre table pour la formule de mise à jour.

5 Formulaires

Rappelons qu'Access est un logiciel de bureautique destiné à la création de formulaires. Tout le travail préparatoire que nous venons de réaliser concernant les tables, leurs clés étrangères et les requêtes, n'est destiné qu'à la création rigoureuse de formulaires et au stockage des données qui sont saisies à travers eux.

Un formulaire est un outil de saisie d'informations (ce ne sont pas des outils d'analyse des informations). Historiquement, les premiers formulaires sont des formulaires papier, comme par exemple : des fiches de renseignements (figure 61), des factures ou des déclaration d'impôt. Aujourd'hui, les formulaires papier existent toujours, mais tendent à être remplacés par des formulaires informatiques, car leur traitement est plus rapide et moins coûteux (au bout d'un certain temps).

Fiche de renseignements

Nom : ..... Prénom : .....
Date de naissance : . . / . . / . . . . Sexe : ☐ F ☐ M
Pays de naissance : ..... Nationalité : .....
Situation familiale[1] : célibataire - marié(e) - séparé(e) - veuf(ve)
Adresse : .....
Code postal : ..... Téléphone : .....

[1] entourer la mention utile

FIG. 61 – *Formulaire papier*

Pendant longtemps, les formulaires informatiques étaient exclusivement en mode texte (sous la forme d'un écran de 25 lignes et de 80 caractères, par exemple). Ce type de formulaire manque cruellement d'ergonomie (l'ergonomie désigne la facilité d'utilisation). Aujourd'hui, il reste énormément de formulaires informatiques en mode texte (essentiellement grâce à leur robustesse), mais il tendent à être remplacés par des formulaires informatiques munis d'une interface graphique (composée de fenêtres, de contrôles et de boutons, comme sur la figure 62).

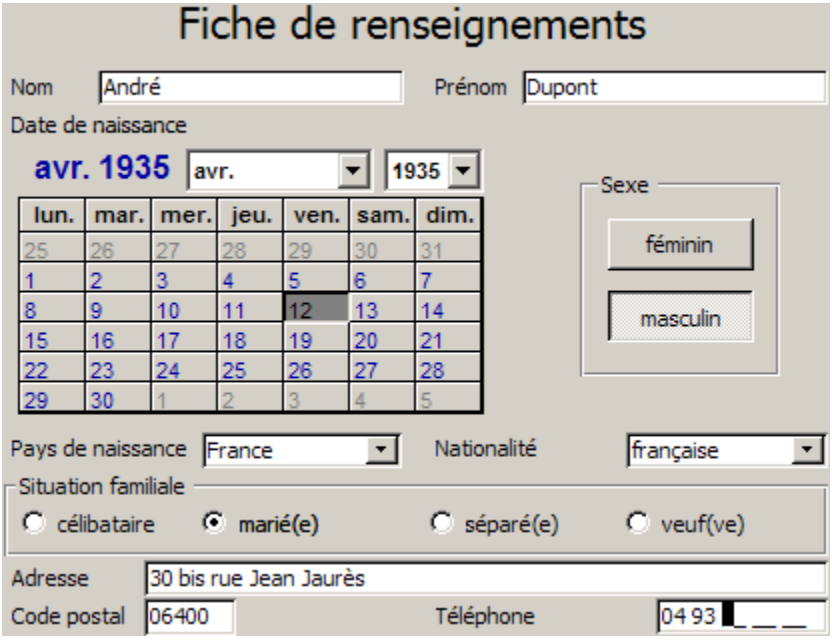


FIG. 62 – *Formulaire informatique avec interface graphique*

5.1 Propriétés d'un formulaire

Nous distinguons quatre types de formulaires réalisables avec Access :

- les formulaires dédiés à la création d'un enregistrement, autrement dit, d'une fiche à la fois (c'est le mode unique) ;
- les formulaires dédiés à la modification des enregistrements existants et pour lesquels nous pouvons faire défiler les fiches (ils sont en mode continu) ;
- les formulaires dédiés à la consultation des enregistrements existants (mode continu, mais modification non autorisée) ;
- et les tableaux croisés dynamiques et les graphiques croisés dynamiques (pour ces notions, nous renvoyons une nouvelle fois le lecteur vers la référence [Isoz]).

Les deux derniers types de formulaires sont dédiés à la recherche ou à l'analyse des données et ne sont pas détaillés dans ce document. Nous nous concentrons sur les formulaires de saisie et les formulaires de modifications.

La distinction entre ces différentes finalités s'établit lors du réglage des propriétés du formulaire (figures 63 et 64).

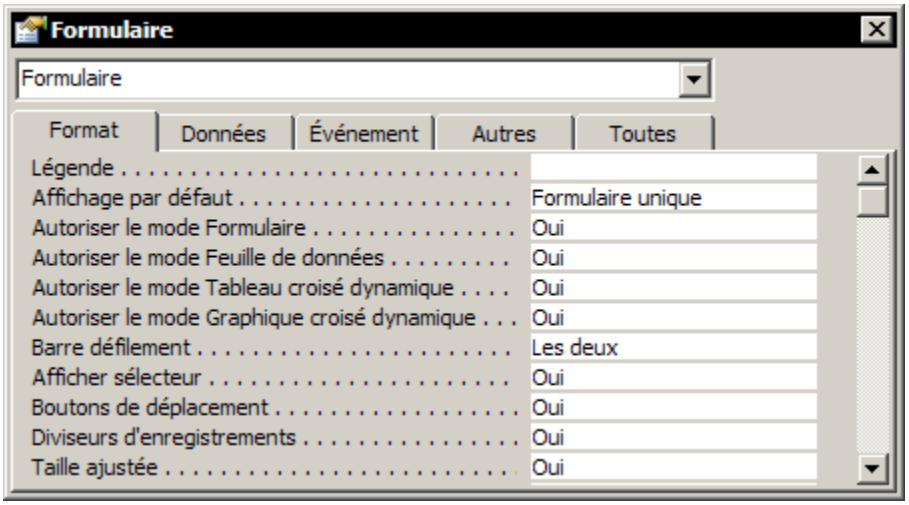


FIG. 63 – *Principales propriétés à régler à la création d'un formulaire (onglet* `Format`*)*

Cette fenêtre des propriétés est disponible en cliquant sur le bouton ou en double cliquant sur le coin supérieur gauche en mode création.

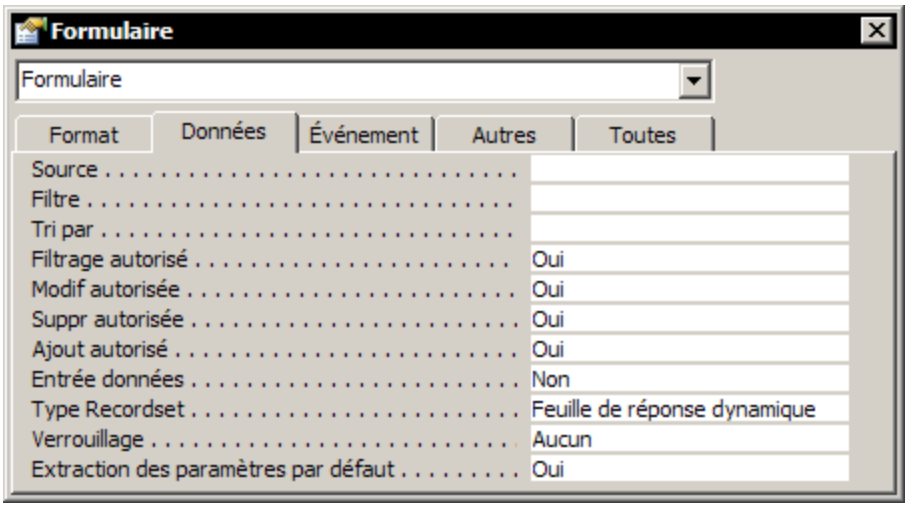


FIG. 64 – *Principales propriétés à régler à la création d'un formulaire (onglet* `Données`*)*

Les valeurs préconisées pour les principales propriétés sont décrites dans le tableau 45.

	formulaires		
propriétés	de saisie	de modification	de consultation
`Affichage par défaut`	unique	continu	continu
`Autoriser le mode Formulaire`	Oui	Oui	Oui
`Autoriser le mode Feuille de données`	Non	Oui	Oui
`Autoriser le mode Tableau croisé dynamique`	Non	Non	Non
`Autoriser le mode Graphique croisé dynamique`	Non	Non	Non
`Afficher sélecteur`	Non	Oui	Oui
`Boutons de déplacement`	Non	Oui	Oui
`Diviseurs d'enregistrements`	Non	Oui	Oui
`Taille ajustée`	Oui	Oui	Oui
`Filtrage autorisé`	Non	Oui	Oui
`Modif autorisée`	Non	Oui	Non
`Suppr autorisée`	Non	Oui	Non
`Ajout autorisé`	Oui	Oui	Non
`Entrée données`	Oui	Non	Non

TAB. 45 – *Réglage des propriétés d'un formulaire selon sa fonction*

Les autres propriétés du formulaire ont un caractère plus anecdotiques ou seront décrites plus loin dans le document.

5.2 Principaux éléments constitutifs

Plaçons-nous en `Mode création` d'un nouveau formulaire, à ne pas confondre avec le mode affichage (c'est-à-dire le `Mode Formulaire`, figure 65) qui permet de visualiser le formulaire créé.

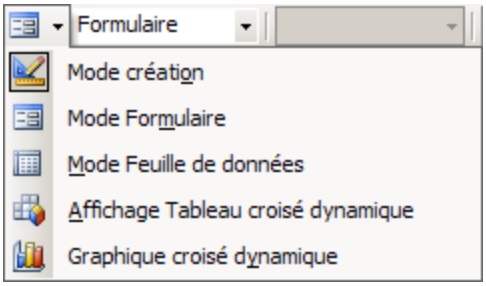


FIG. 65 – *Les différents modes possibles pour un formulaire*

Un formulaire de saisie se décompose en trois zones, dont nous pouvons répartir les rôles ainsi (figure 66) :

- l'en-tête pour y placer le titre du formulaire ;
- le détail, l'endroit où nous organisons la disposition des contrôles ;
- et le pied de formulaire destiné à recevoir les boutons utiles à l'utilisateur [25].

FIG. 66 – *Composition d'un formulaire de saisie*

25. sur la figure 66, se trouvent, de gauche à droite, les boutons `Nouveau`, `Annuler`, `Sauvegarder` et `Quitter`, dont les pictographes ont l'avantage de ne pas être purement francophones, contrairement aux étiquettes

Pour un formulaire de consultation ou de modification ou pour un sous-formulaire, ces trois zones peuvent être utilisées différemment (figure 67) :

- l'en-tête avec, non seulement le titre du formulaire, mais aussi les contrôles qui servent aux critères de sélection et les étiquettes des contrôles de la zone détail ;
- le détail réduit à une ligne qui décrit la structure d'affichage d'une ligne typique (grâce au mode continu, autant de lignes que nécessaire seront affichées à la visualisation du formulaire) ;
- et le pied de formulaire avec, non seulement les boutons appropriés, mais aussi les totaux et les autres informations communes à toutes les lignes qui seront affichées grâce à la zone détail.

FIG. 67 – *Composition d'un formulaire en mode continu*

Les contrôles que nous utilisons dans un formulaire peuvent être classés en trois catégories :

- les contrôles indépendants (les étiquettes et les autres éléments de décoration) qui ont un rôle purement ergonomique ;
- les contrôles dépendants, c'est-à-dire ceux qui permettent de faire le lien entre les données du formulaire et celles des tables (ils ont un rôle essentiel) ;
- et les contrôles calculés, c'est-à-dire ceux qui résultent d'opérations entre plusieurs données (comme un montant total sur une facture, par exemple) ; ceux-là ont un rôle purement informatif.

Chaque contrôle est muni de dizaines de propriétés et l'essentiel du travail pour créer un formulaire consiste à choisir les bons contrôles et de régler leurs propriétés. La gestion des contrôles indépendants est laissée au bon goût du développeur, à condition que l'ergonomie du formulaire ne soit pas sacrifiée pour des raisons esthétiques. Les contrôles calculés et leurs formules sont abordés dans la section 8.2 page 124. Restent les contrôles dépendants, auxquels la section suivante est dédiée.

5.3 Contrôles

Les données saisies dans un formulaire doivent obéir aux mêmes contraintes (types de données, règles de validation, non vacuité et clés étrangères) que les colonnes des tables dans lesquelles elles seront stockées. Le fait de gérer ces contraintes au niveau du formulaire permet d'éviter, à l'utilisateur final, de recevoir des messages d'erreur du moteur Access. Cette exigence conduit à l'élaboration de quatre règles de bonne conduite, à la création d'un formulaire.

5.3.1 Correction du formulaire généré par l'assistant

Règle 1 : à chaque type de données, son type de contrôle (tableau 46).

type de données	contrôle approprié
une chaîne de caractères	une zone de texte prénom jean
une donnée numérique	une toupie (c'est-à-dire une zone de liste, réduite à une ligne) handicap 4
un booléen (`Oui/Non`)	une case à cocher homme ? ☑
une date	un calendrier (il s'agit d'un contrôle ActiveX) juin 2004 juin 2004 lun. mar. mer. jeu. ven. sam. dim. 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 1 2 3 4 5 6 7 8 9 10 11
une heure	une zone de texte avec un masque de saisie heures:minutes:secondes heure 18:12:45

TAB. 46 – *Types de données et contrôles appropriés*

Malheureusement, si nous demandons à Access de générer automatiquement un formulaire à partir de certaines colonnes pré-sélectionnées (avec l'assistant), alors le formulaire ne contient que des zones de texte (bouton ab|) et des cases à cocher (bouton ☑). Il est tout de même conseillé de créer tout nouveau formulaire avec l'assistant. Il suffit simplement de modifier manuellement certains contrôles, afin de se conformer au tableau 46.

Pour l'utilisateur final, le contrôle ActiveX calendrier présente l'avantage d'organiser en semaines les jours du mois choisi. Malheureusement encore, un contrôle ActiveX (que ce soit le calendrier ou non) ne peut pas être utilisé dans le détail d'un formulaire en mode continu (comme un sous-formulaire, par exemple).

Dans ce cas, il doit donc être remplacé par une zone de texte avec le masque de saisie jour/mois/année, accompagné éventuellement d'un bouton qui ouvre une fenêtre contenant le calendrier (comme celui de la figure 112 page 117. Cela réclame cependant un peu de programmation VBA (section 7.2.4).

De même, il existe un contrôle ActiveX pour la toupie, mais il est préférable d'employer une zone de liste pré-remplie et réduite à une ligne, comme suggéré dans le tableau 46.

Règle 2 : les valeurs par défaut doivent être les mêmes entre les colonnes de table et les contrôles correspondants. Idem pour les règles de validation, vide interdit et les masques de saisie (numéro de téléphone, code postal, etc.).

Règle 3 : les clés primaires gérées par numérotation automatique[26] peuvent être affichées, mais ne doivent pas être modifiables (zone de texte verrouillée).

Règle 4 : les clés étrangères doivent être gérées rigoureusement (tableau 47).

cas de figure	contrôle correspondant
une liste immuable de choix connus à l'avance (ce qui est rare)	des boutons radio Saison préférée printemps été automne hiver
une liaison un-à-plusieurs	une liste déroulante terrain: Golf Saint Philippe nom / lieu Jackson Park / Seattle, WA West Seattle / Seattle, WA Jefferson Park / Seattle, WA Parc de la Brague / Biot, France Golf Saint Philippe / Sophia-Antipolis
une liaison plusieurs-à-plusieurs	une zone de liste avec choix multiples participants : Phillips Rene Pipps Georg Pontes Mario Rancé Martine Rodriguez Lino Roel Diego Roulet Annette Rovelli Giovanni

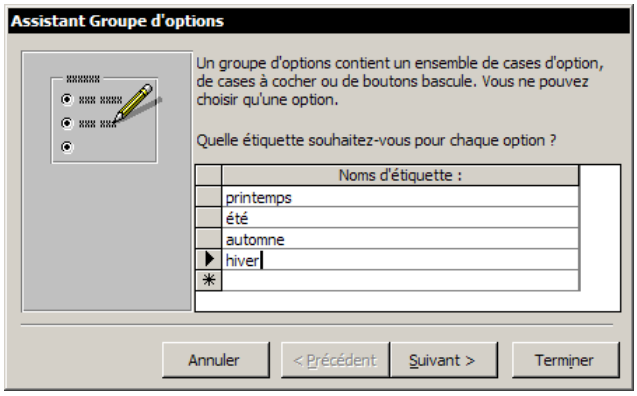
TAB. 47 – *Contrôles appropriés aux différentes clés étrangères*

26. normalement, il s'agit de toutes les clés primaires qui ne sont pas clé étrangère

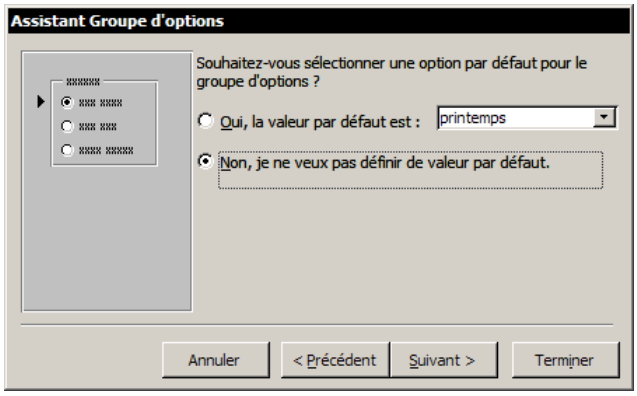
5.3.2 Groupe d'options

Le bouton radio

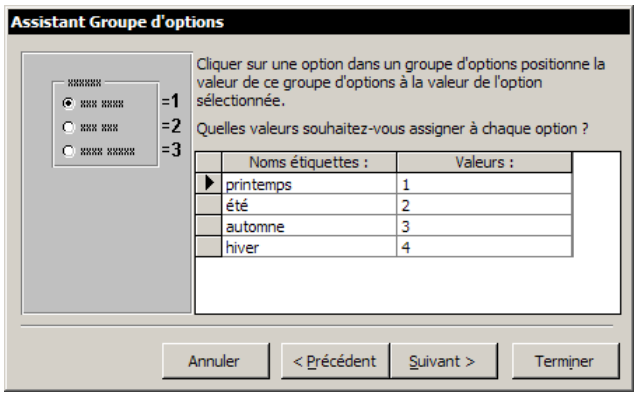
a été créé grâce au bouton `Groupe d'options` qui a ouvert l'assistant `Groupe d'options` (figure 68) car le bouton est enfoncé.



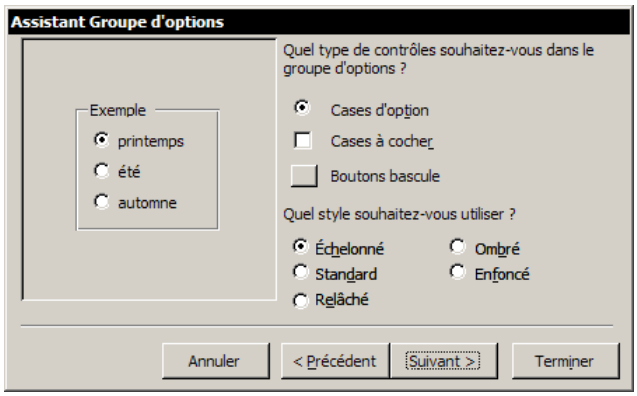
(a) saisie des étiquettes



(b) choix d'une valeur par défaut



(c) choix des valeurs clés



(d) choix de l'apparence

FIG. 68 – *Assistant de création d'un nouveau groupe d'options*

Ce groupe d'options peut prendre d'autres formes que celle du bouton radio. Il peut s'agir d'une case à cocher[27] ou bien d'un bouton bascule (symbolisé par). Cette décision est prise par le développeur du formulaire dans l'assistant (figure 68(d)).

27. mais ce n'est pas conseillé, car cela pourrait induire l'utilisateur en erreur

Les valeurs clés, choisies dans l'assistant (figure 68(c)) peuvent être utilisées ensuite, à condition que la propriété `Source contrôle` du cadre soit positionnée, manuellement, sur une clé étrangère (figure 69).

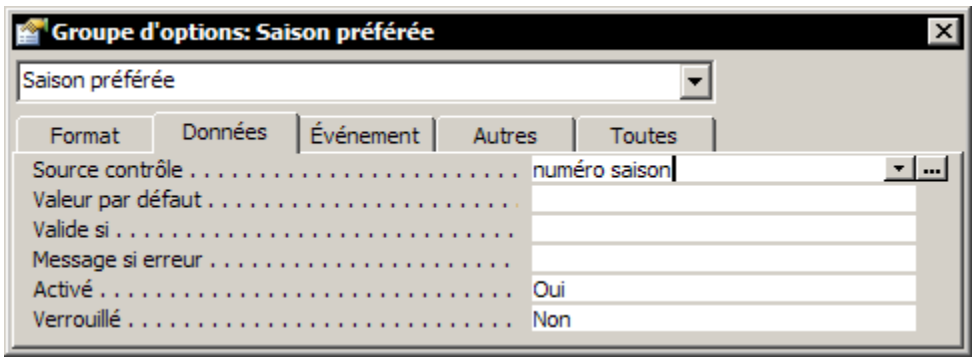


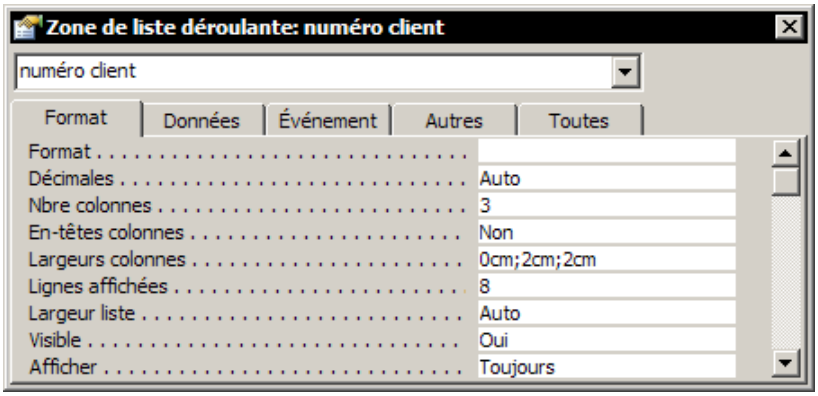
FIG. 69 – *Réglage de la source contrôle d'un groupe d'options sur une clé étrangère*

La clé étrangère `numéro saison` appartient à une table dans la base de données. Mais rien n'oblige le développeur de la base de créer une table spécifique pour stocker les saisons `printemps`, `été`, `automne` et `hiver`, puisque ces valeurs sont précisées manuellement au formulaire qui en a besoin. Toutefois, nous conseillons de créer cette table de codification spécifique pour, d'une part, assurer l'intégrité référentielle de la colonne `numéro saison` et, d'autre part, pouvoir interpréter correctement les valeurs `1`, `2`, `3` et `4` de cette colonne dans un autre formulaire, dans une autre requête ou dans un état.

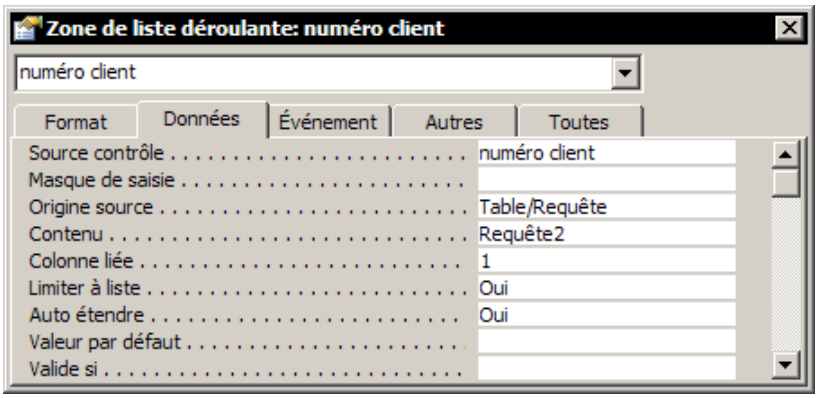
Au final, nous insistons sur le fait que l'utilisation d'un groupe d'options convient très rarement : il faut que la liste de choix soit immuable (dans notre exemple, il est peu probable qu'une cinquième saison apparaissent). *A contrario*, sur le formulaire de la figure 62 page 61, la liste des situations familiales n'est pas immuable (nous avons déjà remarqué qu'il manque la possibilité `pacsé(e)`). Le groupe d'option employé est donc inopportun et il est préférable de le remplacer par une liste déroulante (section suivante).

5.3.3 Liste déroulante

La création d'un formulaire implique la création préalable de plusieurs requêtes qui servent à choisir les tables et les colonnes destinées au stockage des données saisie (propriété `Source contrôle`, figure 70(b)) ou à l'affichage (propriété `Contenu`, même figure).



(a) onglet Format



(b) onglet Données

FIG. 70 – *Principales propriétés à régler à la création d'une liste*

Tout d'abord, à chaque liste du formulaire (liste déroulante, toupie ou tout autre zone de liste) correspond une requête dont le résultat en donne le contenu. La première colonne de cette requête devrait toujours être la clé primaire référencée par la clé étrangère qui correspond à sa liste (afin que la colonne liée de cette liste, soit la colonne 1, dont la largeur est réduite à 0 cm, comme sur la figure 70(a)).

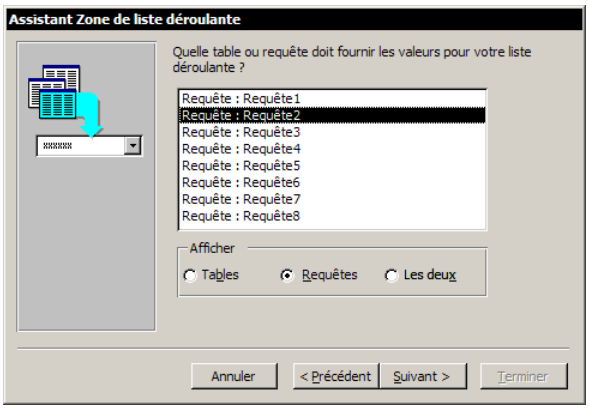
De plus, les intitulés des autres colonnes de cette requête devrait être parfaitement compréhensibles pour l'utilisateur (ce qui n'est pas une exigence pour les intitulés des colonnes de table). Il convient donc de traduire les intitulés (en utilisant une étiquette, c'est-à-dire un alias, comme dans la section 3.5 page 37), le cas échéant.

Par exemple, la requête de la liste déroulante client André Dupuis qui permet de sélectionner le numéro de client (même si ce n'est que son identité qui s'affiche lorsque la liste est repliée) est décrite dans le tableau 48.

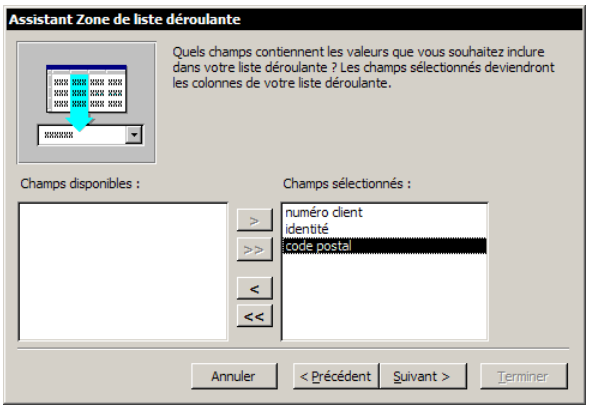
les tables et leurs liaisons	`clients`
les colonnes et leur affichage	`numéro client`, `nom client` (non affiché), `code postal`, `identité: [prénom client] & ' ' & [nom client]`
les critères sur les lignes	aucun
les tris et leur sens	`nom client` (croissant)

TAB. 48 – *Requête-contenu d'une liste déroulante*

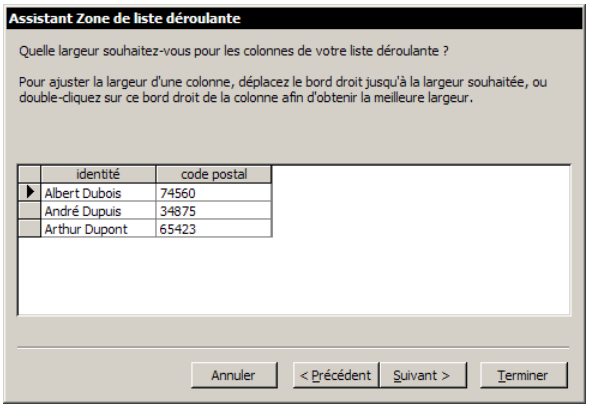
Une fois cette requête-contenu créée, nous pouvons ajouter la liste déroulante correspondante dans le formulaire avec le bouton qui ouvre l'assistant de la figure 71.



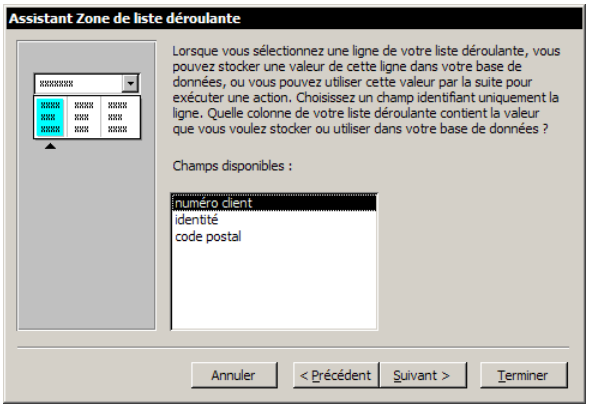
(a) choix de la requête-contenu



(b) choix des colonnes de la liste



(c) réduction de la colonne clé



(d) choix de la colonne clé

FIG. 71 – *Assistant de création d'une nouvelle liste déroulante*

Après avoir choisi toutes les colonnes de la bonne requête (figures 71(a) et 71(b), nous pouvons trier le contenu sur la colonne `identité`, puis réduire la colonne clé à une largeur de `0cm` (figure 71(c)) tout en la choisissant comme colonne liée (figure 71(d)).

Après cet assistant, il reste tout de même à choisir la source contrôle de la liste, c'est-à-dire la clé étrangère qu'elle permet de saisir.

Puis, nous pouvons encore améliorer l'aspect de liste en positionnant à `Oui` sa propriété `En-têtes colonnes` (onglet `Format`), comme sur la figure 72.

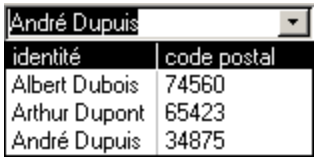


FIG. 72 – *Liste déroulée, avec ses intitulés de colonnes*

D'où l'intérêt de préciser des intitulés clairs et explicites dans la requête du tableau 48.

5.3.4 Toupie par zone de liste

La toupie

que nous préconisons est une zone de liste (bouton) dont la hauteur d'affichage est réduite à une ligne.

Cette zone de liste spéciale exige tout simplement de la part du développeur de la base de données, que soit créée au préalable une table contenant toutes les valeurs numériques possibles pour cette toupie (figure 73).

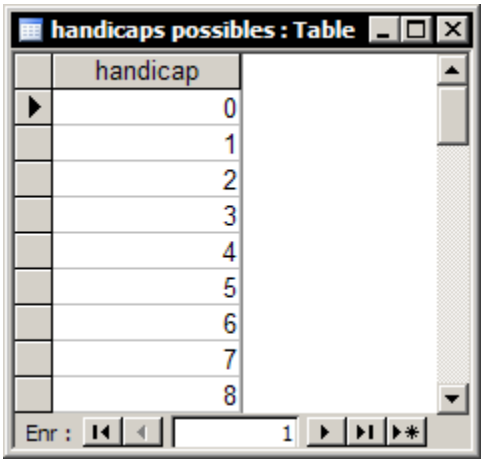
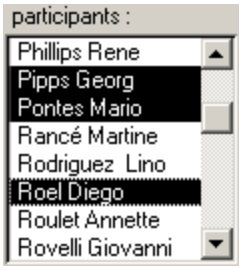


FIG. 73 – *Stockage des valeurs pour une toupie par zone de liste*

Exceptionnellement, cette table n'a pas besoin de figurer sur le schéma relationnel de la base. Cette sert simplement à stocker des valeurs pour la toupie qui en a besoin. En effet, cette table sert ensuite de `Contenu` à la zone de liste qui doit en classer les valeurs dans l'ordre décroissant (à l'aide de l'assistant).

Notons enfin, qu'avec ce type de toupie, il faut tout de même que l'utilisateur du formulaire clique sur la liste afin de valider le choix de sa valeur. En effet, la valeur affichée n'est validée que si elle est surlignée.

5.3.5 Sélection multiple et listes duales

Rappelons qu'une zone de liste telle que 

est créée par le même bouton qui ouvre l'assistant approprié. Cet assistant, qui ressemble beaucoup à l'assistant des listes déroulantes, est très simple à utiliser. Nous conseillons simplement de toujours fonder une telle liste sur une requête écrite au préalable.

La sélection multiple est possible lorsque nous décidons de la valeur de la propriété `Sélection multiple` dans la feuille des propriétés de la zone de liste (figure 74).

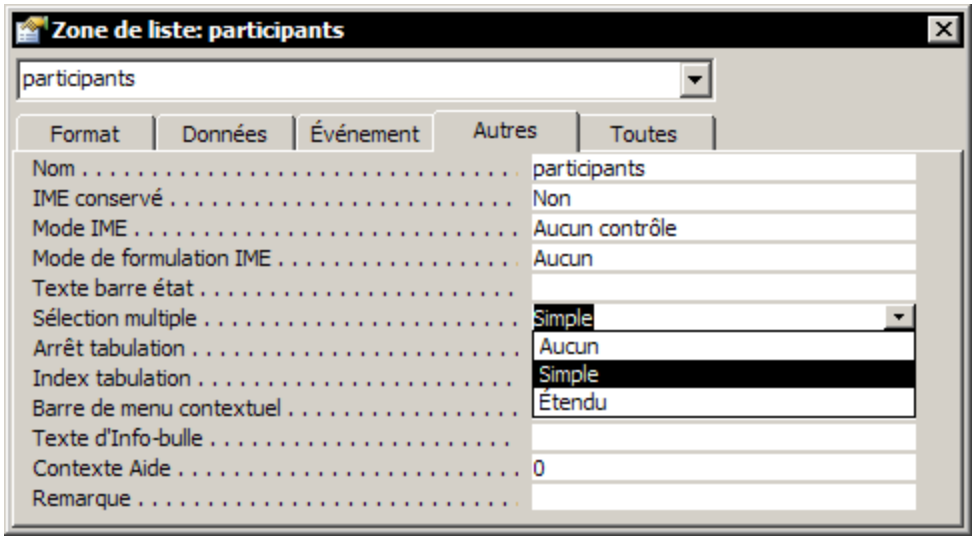


FIG. 74 – *Choix du mode de sélection multiple*

La différence entre le mode `Simple` et le mode `Étendu`, qui sont deux modes de sélection multiple, est laissée à l'expérimentation du lecteur. Il s'agit simplement d'une différence de réaction à l'événement clic sur l'un des éléments de la liste.

Précisons à ce sujet, que les toupies (section précédente) ne doivent pas être à sélection multiple. La valeur de la propriété `Sélection multiple` doit donc être `Aucun`.

D'autre part, pour une liaison plusieurs-à-plusieurs, la zone de liste avec choix multiples peut être remplacée par deux zones de liste, une pour les choix encore possibles et une autre pour les choix retenus, avec deux boutons pour transférer les éléments sélectionnés d'une liste à l'autre (figure 75).

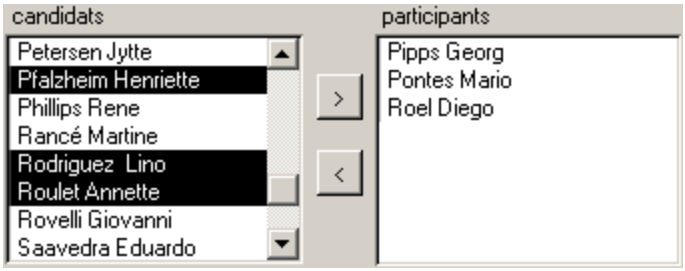


FIG. 75 – *Listes duales avec boutons de transfert*

Quoi qu'il en soit, les zones de liste à choix multiples et les listes duales réclament une programmation VBA spécifique qui est exposée dans la section 8.4 page 129

5.3.6 Compléments sur les listes

Pour une liaison un-à-un, comme il s'agit d'une liaison un-à-plusieurs particulière, le contrôle approprié est aussi une liste déroulante. Par exemple, dans un formulaire de saisie des services, il faut une liste déroulante pour choisir l'employé qui le dirige. Cependant, pour s'assurer qu'un employé ne dirige pas deux services, le contenu de cette liste déroulante doit être calculé *via* une requête afin de ne pas proposer les employés qui dirigent déjà un service.

Pour les auto-liaisons, les liaisons composites et les sous-numérotation, il suffit de déterminer s'il s'agit d'une liaison un-à-plusieurs ou plusieurs-à-plusieurs pour savoir quel contrôle employer. Pour une liaison plusieurs-à-plusieurs-à-plusieurs (ou plus), c'est un sous-formulaire (section suivante) qu'il faudra utiliser.

Dans tous les cas, les listes doivent être les plus ergonomes possible, c'est-à-dire au moins triées dans un ordre logique (alphabétique, par exemple) et en affichant éventuellement plusieurs colonnes (correctement intitulées) afin de bien différencier les éléments de la liste. Par ailleurs, les numéros de clés primaires sont indispensables au traitement des informations saisies pour les clés étrangères à travers ces listes, mais ils doivent être masqués à l'utilisateur qui n'a pas besoin de les connaître (en réduisant à zéro la largeur de leur colonne).

Enfin, il ne doit pas être permis à l'utilisateur du formulaire de saisir une autre valeur que celles proposées dans les listes. Pour cela, il faut positionner à `Oui` la propriété `Limiter à liste` de l'onglet `Données`, comme sur la figure 70(b) page 70.

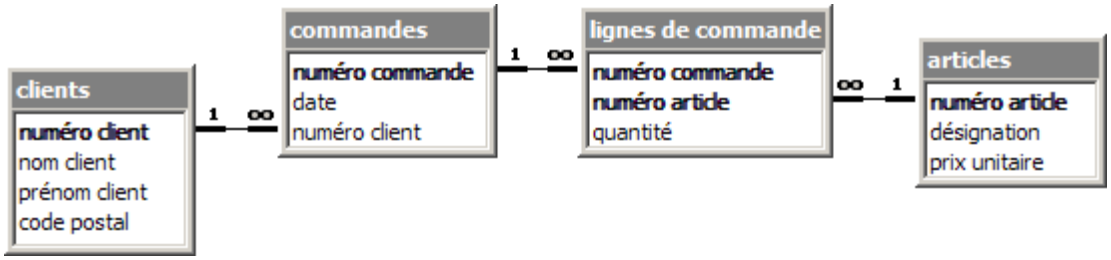
Pour parer au cas où l'élément recherché par un utilisateur n'existe pas encore, le formulaire peut proposer un bouton qui ouvre un autre formulaire de saisie afin de créer cet élément (cette fonctionnalité est décrite dans la section 7.2.4). Une autre possibilité consiste à profiter du fait que l'utilisateur peut saisir au clavier la valeur voulue, directement dans la liste, pour récupérer cette saisie et en offrir l'insertion automatique dans la table concernée (pour cela il faut programmer un bout de code VBA, comme dans la section 8.6 page 138).

Mais cette deuxième possibilité ne peut fonctionner que si la table ne contient qu'une colonne (en plus de sa clé primaire), typiquement, une table de codification (comme sur la figure 3(b) page 6). Pour une liste déroulante reposant sur une table ayant plusieurs colonnes à renseigner, il faut ouvrir le formulaire de saisie spécifique.

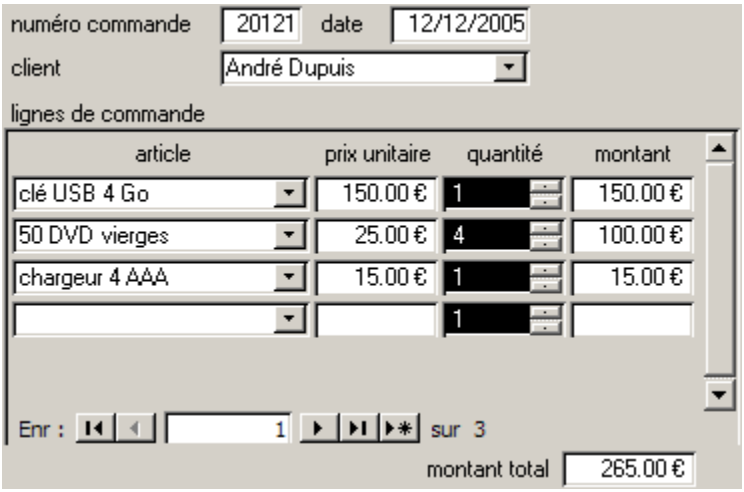
5.4 Sous-formulaire

Pour une liaison plusieurs-à-plusieurs, les zones de liste, même duales, ne permettent pas de saisir les autres colonnes de la table de jonction. Par exemple, dans un formulaire de saisie d'une commande, avec une zone de liste à sélection multiple, nous pourrions choisir les articles commandés, mais pas leur quantité commandée, car une zone de liste ne permet que le choix d'une clé étrangère.

Pour pouvoir choisir un ou plusieurs articles et saisir leur quantité respective, nous employons un sous-formulaire (figure 76). Cela permet, de surcroît, de laisser affichés l'intitulé et le prix unitaire, de saisir la quantité puis d'afficher le montant (par un contrôle calculé dont la formule, simple, consiste à multiplier le prix unitaire par la quantité).



(a) schéma relationnel



(b) formulaire de facturation

FIG. 76 – *Utilisation d'un sous-formulaire*

Le sous-formulaire (bouton) est un formulaire en mode continu et se compose :

- d'un en-tête, dans lequel sont placés en ligne, les intitulés des colonnes du détail ;
- d'un détail réduit à une ligne de contrôles (cette ligne sera répétée autant de fois que nécessaire en mode affichage) ;
- d'un pied de formulaire, vide sur la figure 76(b), car le montant total a besoin du numéro de commande et est donc placé dans le pied du formulaire principal.

Le formulaire de la figure 76(b) a été créé à l'aide de l'assistant dont la deuxième étape, cruciale, permet de choisir le mode sous-formulaire pour gérer les lignes de commandes (figure 77).

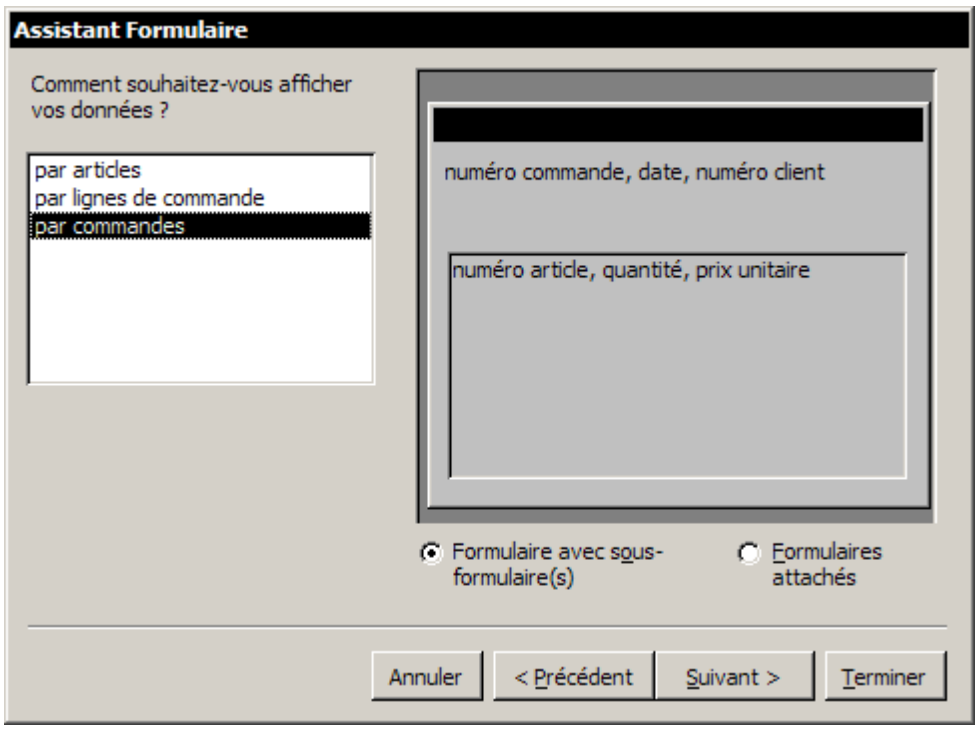


FIG. 77 – *Choix du mode sous-formulaire dans l'assistant de création d'un formulaire*

Le formulaire de la figure 76(b) permet de saisir à la fois une nouvelle commande et toutes ses lignes de commandes, autrement dit, des données réparties dans deux tables distinctes (figure 76(a)). La table `articles` est utilisée pour afficher le prix unitaire, mais comme ce contrôle est verrouillé[28], aucune données n'est finalement saisie dans cette table `articles`. De même, les informations de la table `clients` ne sont utilisées que pour saisir la clé étrangère de la table `commandes`.

Il importe donc de déterminer, avant de créer un nouveau formulaire, quelles tables serviront de stockage et quelles tables serviront d'affichage[29]. Cette distinction se retrouve entre les propriétés `Source contrôle` et `Contenu` (section suivante).

Mais auparavant, nous insistons sur le fait qu'un sous-formulaire ne permet pas de saisir de données dans la troisième table (ici la table `articles`) en liaison plusieurs-à-plusieurs par rapport à la table du formulaire principal. Cette troisième table en liaison plusieurs-à-plusieurs doit être remplie au préalable[30].

Au final, un formulaire muni d'un sous-formulaire, ne permettent de saisir des données que dans deux tables liés par une liaison un-à-plusieurs (dans l'exemple de la figure 76, il s'agit de la liaison entre la table `commandes` et sa table de jonction `lignes de commandes`), même si la table `articles` figure dans la requête source pour afficher le prix unitaire.

28. l'utilisateur ne peut pas changer le prix unitaire et c'est normal
29. pour un formulaire donné, certaines tables peuvent servirent à la fois de stockage et d'affichage
30. ou éventuellement complétée au besoin, au travers d'un autre formulaire disponible *via* un bouton (pour cette fonctionnalité, nous renvoyons à nouveau le lecteur vers la section 7.2.4 page 116)

5.5 Requête source

Rappelons que la création d'un formulaire requiert l'écriture préalable de toutes les requêtes-contenu de ses listes, afin d'en remplir la propriété `Contenu`.

À toutes ces requêtes de listes, il est conseillé d'ajouter la création de la requête du formulaire, autrement dit, la requête qui sélectionne les colonnes dans lesquelles seront stockées les données saisies par l'utilisateur. Il s'agit en réalité de toutes les colonnes nécessaires aux propriétés `Source contrôle` des contrôles dépendants.

Le résultat de cette requête source devrait être trié s'il s'agit d'un formulaire en mode continu, mais les intitulés de ses colonnes n'ont pas besoin d'être explicités car seul le développeur du formulaire s'en servira. À titre d'illustration, la requête du formulaire de la figure 76(b) est donnée par le tableau 49.

les tables et leurs liaisons	`commandes`, `lignes de commande`, `articles` (ces tables sont correctement liées par le schéma relationnel)
les colonnes et leur affichage	`commandes.[numéro commande]`, `commandes.[numéro client]`, `date`, `[lignes de commande].[numéro article]`, `quantité`, `articles.[prix unitaire]`
les critères sur les lignes	aucun (inutile pour un formulaire de saisie)
les tris et leur sens	aucun (inutile pour un formulaire de saisie)

Tab. 49 – *Requête source d'un formulaire*

C'est cette requête source qui permet à l'assistant de la figure 77 de détecter automatiquement l'opportunité d'organiser un sous-formulaire pour les lignes de commandes. L'écriture de cette requête est donc une aide précieuse au développement rapide du formulaire de la figure 76(b).

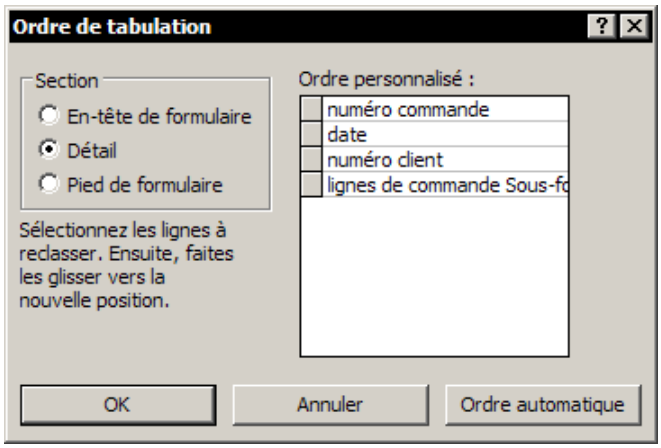
Concernant les contrôles calculés, qui sont généralement des zones de texte, la propriété `Source contrôle` ne contient pas le nom de la colonne de stockage puisqu'aucune donnée ne sera saisie dans ce contrôle purement informatif. Au contraire, cette propriété `Source contrôle` contient la formule du contrôle calculé. C'est ce qui différencie les contrôles calculés des contrôles dépendants.

L'exemple de la figure 76(b) l'illustre parfaitement, puisque le montant est calculé à l'aide de la formule `quantité * [prix unitaire]` que le développeur écrit directement dans la propriété `Source contrôle` de la zone de texte concernée (dans le détail du sous-formulaire). Quant au montant total, il est calculé grâce à la fonction `SomDom`, qui est décrite dans la section 8.2 page 124.

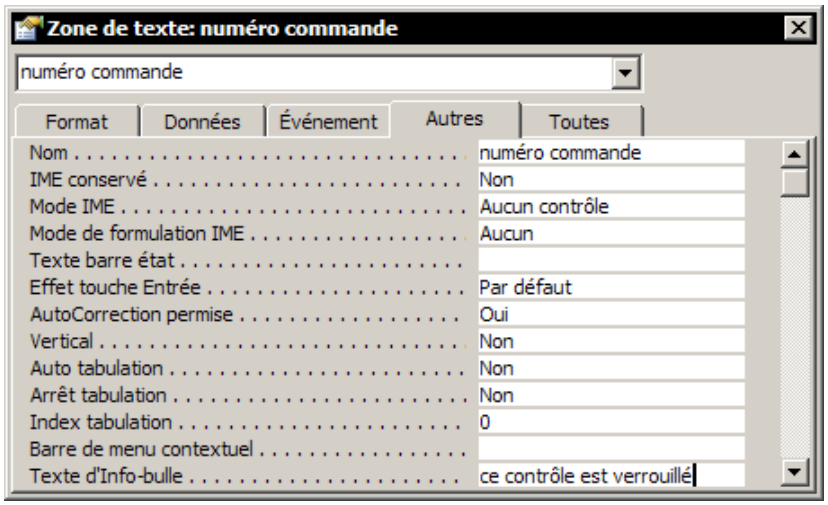
5.6 Ergonomie

Dans un formulaire, les règles ergonomiques suivantes sont dictées par le bon sens :

- un formulaire ne doit pas dépasser la taille de l'écran, en largeur et en hauteur (si nécessaire nous pouvons recourir aux sauts de page, bouton , et aux onglets, bouton) ;
- les contrôles doivent être suffisamment larges pour en lire le contenu ;
- les contrôles successifs à renseigner, doivent être disposés dans un ordre logique (de haut en bas et de gauche à droite) ;
- les tabulations (touches `TAB` et `SHIFT + TAB`) doivent suivre la même logique ;
- les contrôles verrouillés doivent être retirés de la liste des tabulations (propriété `Arrêt tabulation = Non`, figure 78(b)) ;
- tandis que les boutons doivent figurer dans le pied du formulaire, pour apparaître dans la liste des tabulations de cette zone (figure 78(a)).



(a) boîte de dialogue pour les tabulations



(b) propriétés tabulation

FIG. 78 – *Deux façon de gérer les tabulations*

Les tabulations peuvent être organisées grâce à la boîte de dialogue de la figure 78(a), disponible en mode création par le menu `Affichage`, `Ordre de tabulation...`. L'arrêt et l'ordre de tabulation

d'un contrôle peuvent également être modifiés dans la fenêtre des propriétés du contrôle : il s'agit des propriétés `Arrêt tabulation` et `Index tabulation` de l'onglet `Autres` (figure 78(b)).

Les contrôles verrouillés (notamment, tous les numéros automatiques et tous les contrôles calculés), peuvent précisément l'être en positionnant à `Oui` la propriété `Verrouillé` de l'onglet `Données` (figure 79). Il convient cependant d'en informer l'utilisateur par un texte d'info-bulle explicite (figure 78(b), dernière rangée). De surcroît, le fait de positionner à `Non` la propriété `Activé` (figure 79), empêche l'utilisateur d'entrer dans le contrôle verrouillé (sans empêcher l'affichage de l'info-bulle).

FIG. 79 – *Verrouillage et désactivation d'un contrôle*

Il également possible d'activer les contrôles d'un formulaire les uns après les autres, pour obliger l'utilisateur à saisir les informations dans l'ordre, mais c'est souvent inutile car l'utilisateur peut[31] revenir en arrière. Quant à la solution qui consiste à décomposer un formulaire comportant plusieurs questions, en autant de formulaires ne comportant qu'une question et ouverts successivement, elle n'est pas conseillée car le passage des informations d'un formulaire à l'autre est ardue.

Par ailleurs, les contrôles sont des objets qui réagissent à des événements, comme par exemple :
- l'entrée dans un contrôle (par le clavier ou la souris) ;
- la sortie d'un contrôle ;
- la modification de la valeur d'un contrôle.

À ces événements, nous pouvons associer des actions, dont :
- l'activation ou la désactivation d'un autre contrôle ;
- la mise à jour d'une liste ou d'un contrôle calculé ;
- l'affichage d'un message d'erreur explicite en cas de saisie erronée.

Grâce à ces actions événementielles, l'ergonomie d'un formulaire peut être encore améliorée :
- dans une fiche de renseignements, le sous-formulaire concernant les enfants peut être activé ou désactivé, selon que le nombre d'enfant saisi est > 0 ou nul ;
- la liste des jours peut être recalculée selon que le mois sélectionné précédemment est un mois à 30 jours, 31 jours ou le mois de février.

La programmation de ces procédures en VBA, fait l'objet du chapitre 7 page 100.

31. ou devrait toujours pouvoir

5.7 Recherche, tri et filtre

Dans un formulaire de modification ou de consultation des données, l'utilisateur navigue entre les fiches (par opposition à un formulaire de saisie qui ne manipule qu'une nouvelle fiche). Pour faciliter cette navigation, l'utilisateur peut à tout moment :

- rechercher la première fiche dont un champ vérifie un critère (par exemple, le premier livre dont le titre contient le mot `Soleil`) à l'aide du bouton qui ouvre la boîte de dialogue de la figure 80 ;
- trier les fiches dans l'ordre croissant ou décroissant d'un des champs du formulaire (par exemple, trier les livres dans l'ordre alphabétique de leur titre) en cliquant sur le champ puis sur le bouton ou sur le bouton ;
- définir un filtre sur un ou plusieurs champs du formulaire puis appliquer ou non ce filtre sur les fiches du formulaire à l'aide du bouton .

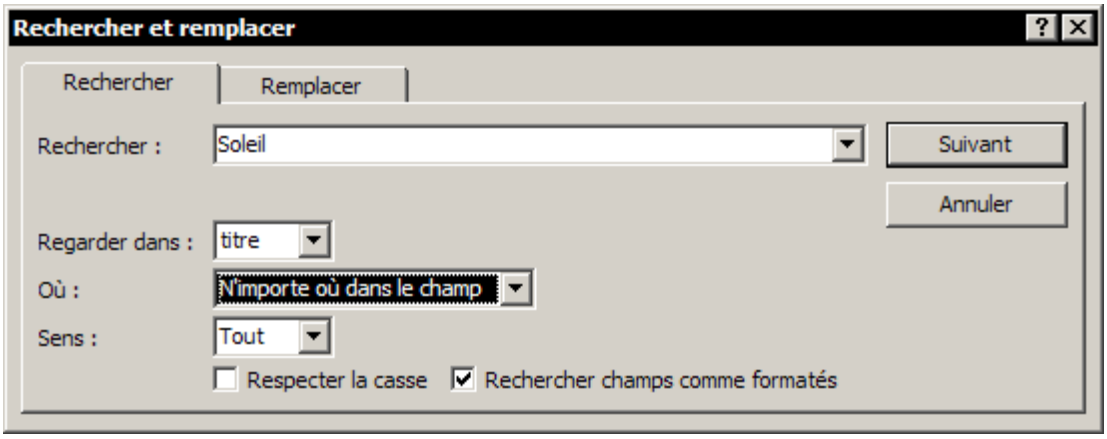


FIG. 80 – *Boîte de dialogue de la recherche parmi les fiches d'un formulaire*

L'utilisateur peut définir un filtre au travers du formulaire vidé de ses données (figure 81) grâce au bouton `Filtrer par formulaire` .

FIG. 81 – *Filtre défini à travers un formulaire de saisie converti en formulaire de consultation*

Cela évite d'avoir à développer un nouveau formulaire pour la consultation, puisqu'avec cette fonctionnalité, il suffit d'utiliser le formulaire de saisie correspondant (ou du moins une copie dont les propriétés sont réglées pour la consultation). Il reste à choisir des valeurs pour les contrôles voulus[32].

Notons que sur l'exemple de la figure 81, seront sélectionnés par le filtre, tous les golfeurs dont le prénom commence par `Mari` (nous pouvons utiliser toutes les possibilités offertes par les critères dans les requêtes, comme dans la section 3.6 page 38), quelque soient leurs nom, handicap et genre. Nous pouvons par ailleurs réunir les résultats de plusieurs filtres, au sein d'un même filtre, avec les onglets `Ou` qui apparaissent en bas.

Il est également possible de définir à la fois les tris et les filtres portant éventuellement sur plusieurs colonnes, à l'aide de l'outil `Filtre/tri avancé` (bouton). Mais cela revient exactement à décrire les critères de sélection d'une requête. Cette fonctionnalité est donc réservée aux utilisateurs familiarisés avec Access.

5.8 Conclusion

La méthodologie que nous préconisons dans l'élaboration des formulaires, consiste à créer d'abord les formulaires de saisie, en commençant par les plus simples, à savoir ceux qui correspondent aux tables sans clé étrangère. Il est alors possible de convertir une copie de chacun de ces formulaires de saisie, en formulaire de modification, puis en formulaire de consultation.

La procédure recommandée pour établir un nouveau formulaire de saisie est la suivante :

1. le dessiner, afin de décider de ses fonctionnalités et de son organisation ;
2. créer les requêtes nécessaires aux listes déroulantes et aux zones de liste (y compris les toupies) ;
3. créer la requête source du formulaire lui-même ;
4. créer le formulaire, à l'aide de l'assistant ;
5. régler les propriétés du formulaire ;
6. convertir les contrôles inappropriés, ajouter les contrôles manquants et régler leurs propriétés respectives.

32. par contre, une valeur saisie dans un sous-formulaire ne peut pas filtrer les fiches du formulaire principal, contrairement à ce que nous aurions pu attendre

6 États

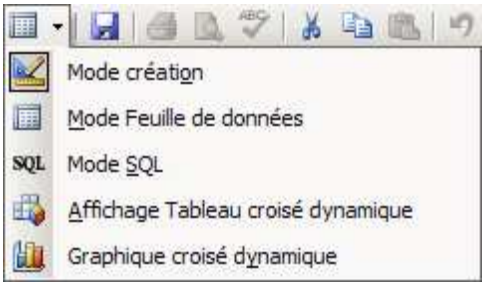
Une base de données Access est à la fois un outil pour recueillir des informations, comme nous venons de le voir avec les formulaires, mais aussi pour les restituer à l'utilisateur afin qu'il puisse les analyser. Pour cela, le logiciel offre la possibilité d'éditer des états, qui sont des documents de synthèse établis automatiquement à partir des données.

À titre d'exemples, considérons les états suivants :

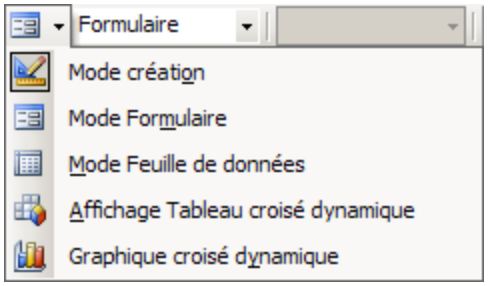
- un annuaire peut être calculé automatiquement à partir des colonnes nom, prénom, numéro de téléphone, adresse, code postal, ville et pays ; il s'agit d'un état purement informatif ;
- une série d'étiquetages peut également être créée automatiquement à partir des données et constitue un état [33] ;
- un tableau de bord des ventes de la veille, ventilées selon les catégories de produits et en comparaison avec la même date l'année précédente, constitue un état à caractère décisionnel ;
- un graphique de répartition peut aussi être considéré comme un état à caractère décisionnel.

Notons à cet occasion que si Access propose les modes d'affichage `Tableau croisé dynamique` et `Graphique croisé dynamique`, à la fois pour les tables, les requêtes et les formulaires (ou plutôt, pour la requête source des formulaires), il s'agit en fait du même service (figure 82).

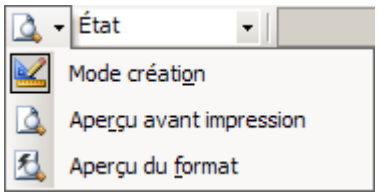
(a) pour les tables



(b) pour les requêtes



(c) pour les formulaires



(d) pour les états

FIG. 82 – *Les différents modes d'affichage des objets Access*

Il est d'ailleurs surprenant, que ce service ne soit pas proposé pour les états (ou plutôt, pour leur requête source), alors même qu'il s'agit de deux services analytiques proches.

33. mais cette fonctionnalité est plus riche avec le logiciel Word ; nous recommandons d'ailleurs d'utiliser le publipostage de Word, tout en laissant les données stockées dans Access

6.1 Création

Les états, comme les formulaires, reposent sur une requête préalable qui sélectionne les données utiles, mais les états sont fondamentalement différents des formulaires. En effet, contrairement aux formulaires, les états ne permettent pas à l'utilisateur de modifier les données, mais simplement de les agréger puis de les consulter. En conséquence, les états sont aussi composés de contrôles, mais essentiellement de zones de texte, ainsi que de leurs étiquettes.

De plus, les formulaires permettent, certes, d'imprimer les données (une facture, par exemple), mais les états sont destinés à être imprimés. Nous veillerons donc particulièrement à ce que :

- la largeur de l'état ne dépasse pas celle d'une feuille (format A4, généralement, c'est-à-dire 21 centimètres), que nous pouvons éventuellement orienter `Paysage`, afin de gagner en largeur ;
- les contrôles soient suffisamment larges pour en afficher le contenu[34] ;
- l'état contienne toutes les informations nécessaires à sa compréhension et, surtout, les conditions de sélection de la requête sur laquelle il repose.

La taille des pages d'un état se décide, comme pour tout document Office, dans le menu `Fichier`, `Mise en page...` qui permet de choisir la taille du papier, son orientation et la largeur des marges (figure 83).

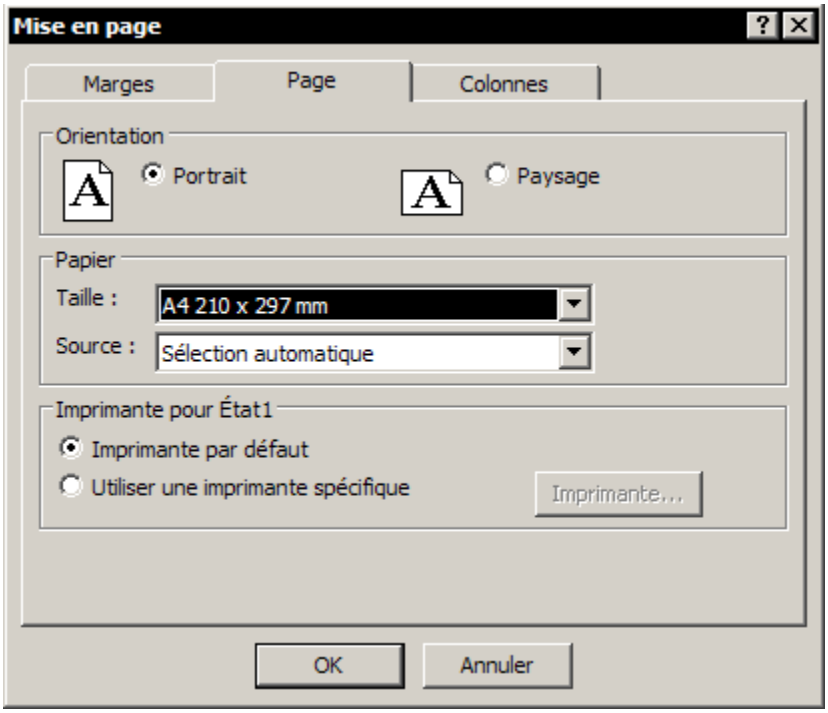


Fig. 83 – *Choix de la mise en page (c'est crucial pour un état)*

La première étape de création d'un état est la même que pour un formulaire, puisqu'il s'agit de le dessiner pour en organiser la structure (section suivante).

34. c'est encore plus impératif pour un état imprimé que pour un formulaire informatique, car un formulaire permet tout de même de révéler les caractères cachés (à condition de deviner qu'il y en a)

6.1.1 Composition

La composition d'un état est plus riche que celle d'un formulaire, car elle offre, en plus de la zone détail, plusieurs en-tête et plusieurs pieds qui ont chacun un rôle précis à jouer (figure 84).

État des commandes par article et par client				← en-tête d'état : le titre et les conditions de sélection (indispensables pour comprendre les quantités)
pour les commandes passées hier				
(les prix et les montants sont en euros)				← en-tête de page : les unités (c'est important) et les intitulés du détail
	client	quantité	montant	
50 DVD vierges	prix : 25.00			← en-tête de groupe : infos du groupe
	Albert Dubois	2	50.00	← zone détail : une ligne par sous-groupe
	André Dupuis	4	100.00	
	Total	6	150.00	← pied de groupe : totaux intermédiaires
clé USB 4 Go	prix : 150.00			← en-tête recalculé à chaque changement de groupe
	Arthur Dupont	1	150.00	
	André Dupuis	1	150.00	
	Total	2	300.00	
	page 1 / 2			← pied de page : numéro de page

(les prix et les montants sont en euros)				
	client	quantité	montant	← en-tête répété à chaque page
chargeur 4 AAA	prix : 15.00			
	Albert Dubois	1	15.00	
	Arthur Dupont	2	30.00	
	André Dupuis	1	15.00	
	Total	4	60.00	
Grand Total		12	510.00	← le pied d'état : totaux généraux
	page 2 / 2			← utile en cas de mélange des feuilles imprimées

FIG. 84 – *Description des différentes zones d'un état, sur une composition manuelle*

Cet état est volontairement schématique, afin de bien visualiser les différentes composantes. Il est, dans la suite, créé étape par étape avec Access.

6.1.2 Requête source

Comme pour un formulaire, il est vivement recommandé de créer un nouvel état à l'aide de l'assistant. Pour cela, il faut au préalable écrire la requête sur laquelle cet état repose, c'est-à-dire la requête qui sélectionne les colonnes nécessaires au(x) niveau(x) de regroupement, à l'affichage et au(x) tris. Pour l'état de la figure 84, il s'agit de la requête du tableau 50.

les tables et leurs liaisons	`clients`, `commandes`, `lignes de commande`, `articles` (ces tables sont correctement liées par le schéma relationnel de la figure 76(a))
les colonnes et leur affichage	`[lignes de commande].[numéro article]` (non affichée), `commandes.[numéro client]` (non affichée), `[lignes de commande].quantité`, `montant: [lignes de commande].quantité * articles.[prix unitaire]`, `identité: [prénom client] & ' ' & [nom client]`, `[nom client]`, `désignation`, `[prix unitaire]`, `date`
les critères sur les lignes	`date = AjDate("j";-1;Date())`
les tris et leur sens	`nom client` (croissant)
les opérations	`Regroupement` sur `[lignes de commande].[numéro article]`, `Regroupement` sur `commandes.[numéro client]`, `Somme` sur `[lignes de commande].quantité`, `Somme` sur `montant`, `Premier` sur `identité`, `Premier` sur `[nom client]`, `Premier` sur `désignation`, `Premier` sur `[prix unitaire]`, `Où` sur `date`,

TAB. 50 – *Requête d'un état, avec pré-agrégation*

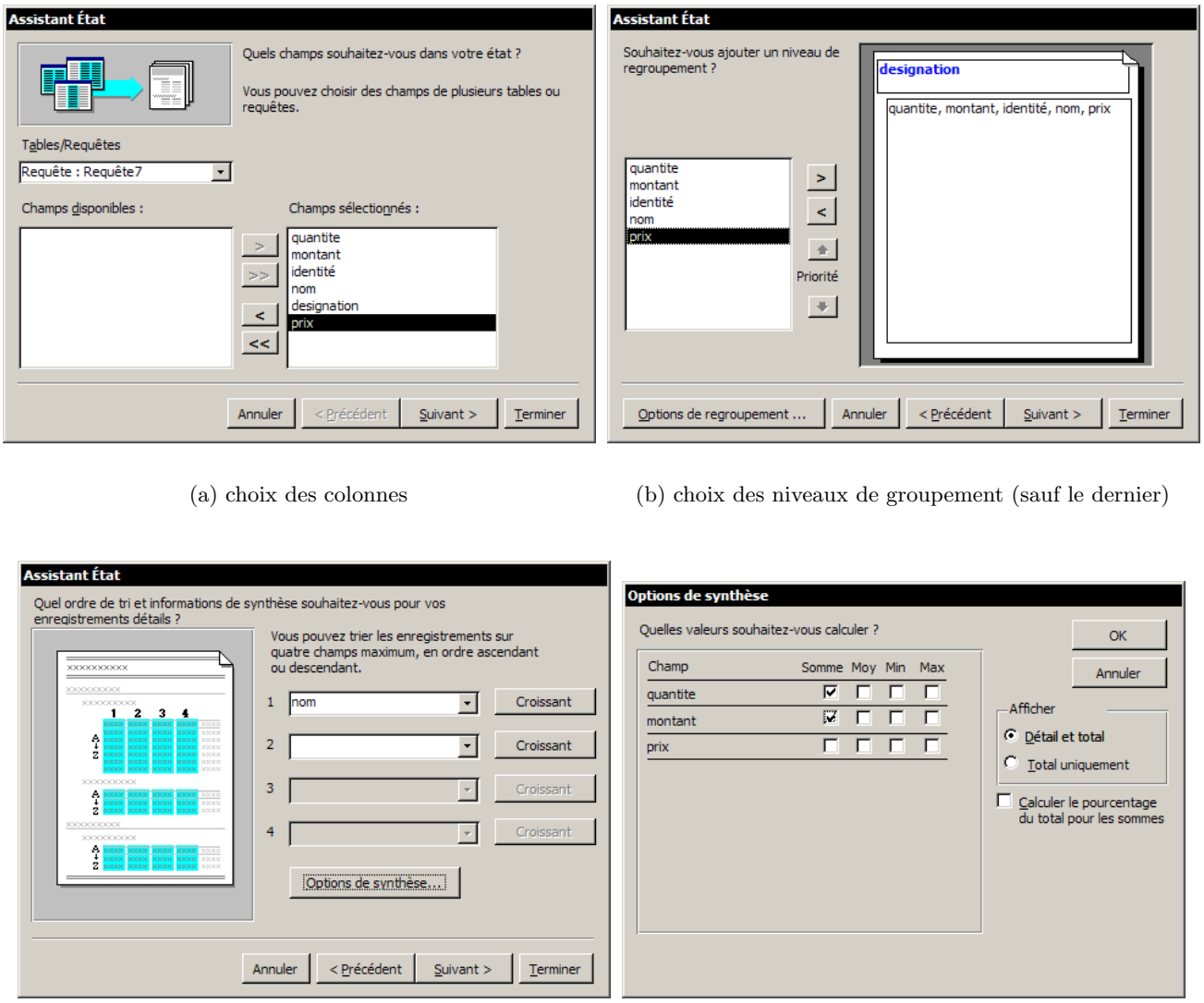
Comme souvent avec les états, cette requête fait intervenir des regroupements (auxquels nous avons consacré la section 3.7 page 40), le résultat de cette requête est donc pré-agrégé. Les colonnes de regroupement correspondent aux niveaux de groupement désirés dans l'état, y compris le groupe correspondant au détail. L'état de la figure 84 repose sur deux niveaux de groupement : d'abord les articles, puis les clients (niveau détail). C'est pour cette raison que les deux colonnes de regroupement de la requête sont le `numéro article` et le `numéro client`. Ces valeurs ne seront pas affichées par l'état, nous ne les affichons donc pas non plus dans la requête.

Ensuite, nous ajoutons à la requête les colonnes sur lesquelles les opérations d'agrégation (somme, moyenne, minimum ou maximum) seront effectuées dans l'état et nous leur affectons la fonction de pré-agrégation convenable (le plus souvent, il s'agit d'un dénombrement ou d'une somme).

Puis nous ajoutons les informations complémentaires et les colonnes de tri (qu'il faut afficher), avec la fonction d'agrégation `Premier` (ou `Dernier`) et les mêmes tris que dans l'état (mais ce n'est pas obligatoire). Sans oublier, enfin, les colonnes pour les critères de sélection (qu'il faudra décrire dans l'entête d'état), avec la fonction d'agrégation `Où`. Ici, pour calculer la date d'hier, nous utilisons la fonction `Date()` pour obtenir la date d'aujourd'hui et la fonction `AjDate` pour lui retrancher un jour.

6.1.3 Utilisation de l'assistant

Une fois la requête écrite (et consultée en mode affichage pour contrôler sa validité), le nouvel état peut être créé à l'aide de l'assistant. La première étape consiste à sélectionner les colonnes utiles à l'état. Pour nous, il s'agit simplement de toutes les colonnes de la requête précédente (figure 85(a)).



(a) choix des colonnes

(b) choix des niveaux de groupement (sauf le dernier)

(c) choix du tri

(d) choix des agrégations

FIG. 85 – *Création d'un état avec l'assistant*

L'étape suivante consiste à choisir les niveaux de groupement dans le bon ordre, sauf le dernier niveau qui correspond au niveau détail. Pour nous qui avons deux niveaux : d'abord les articles, puis les clients (niveau détail), il suffit d'ajouter le niveau qui repose sur la colonne désignation (figure 85(b)).

Ensuite il faut choisir les tris et préciser leur sens (croissant ou décroissant). Dans notre exemple, nous n'avons qu'un seul tri à préciser : celui qui porte sur la colonne `nom` afin que les clients soient classés par ordre alphabétique de leur nom de famille (figure 85(c)).

Puis, en cliquant sur le bouton `Options de synthèse...`, nous pouvons préciser sur quelles colonnes nous souhaitons voir apparaître un sous-total et un total global. Pour notre état, nous souhaitons que la somme des quantités et la somme des montants soit effectuées (figure 85(d)).

Il reste enfin à choisir la disposition et le titre souhaités, avant de terminer l'assistant.

6.1.4 Correction de l'état obtenu

Nous obtenons alors, en mode création, l'état de la figure 86.

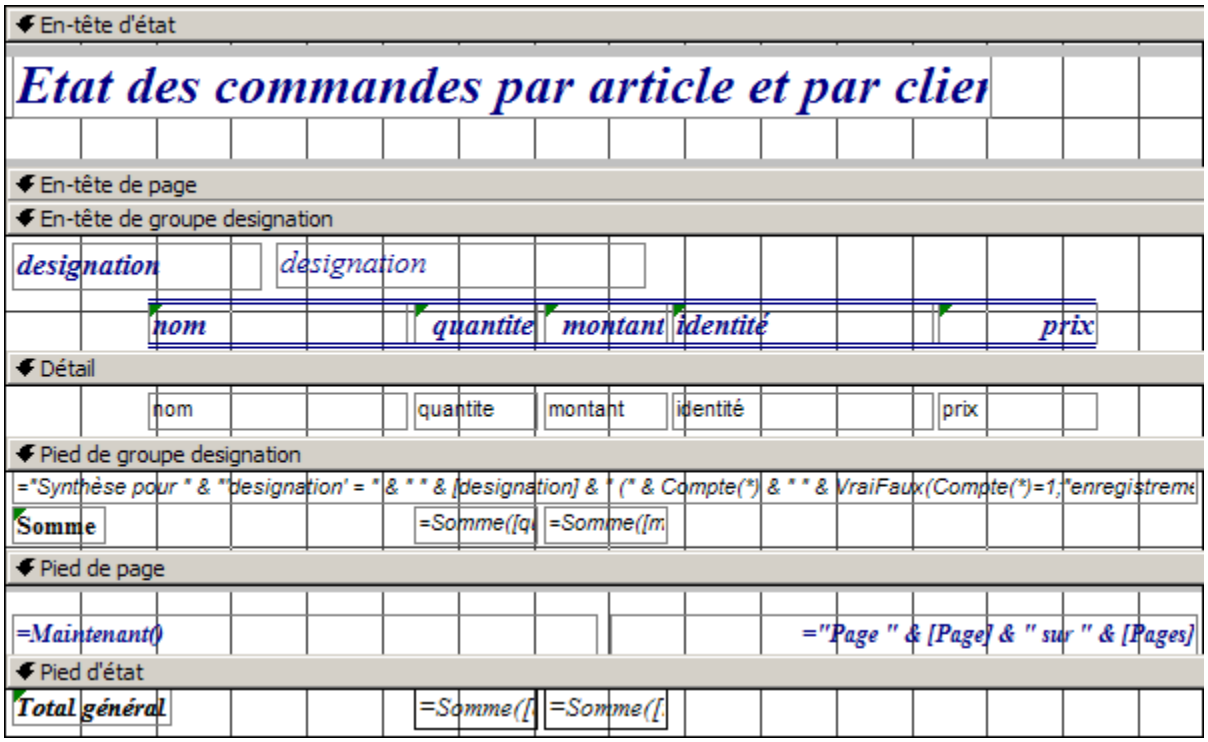


FIG. 86 – *État généré par l'assistant avant correction*

Cet état ne convient pas parfaitement car :
- certains contrôles ne sont pas suffisamment larges ;
- il manque les conditions de sélection ;
- il manque les unités ;
- les intitulés du détail ne sont pas dans la bonne section ;
- certaines étiquettes et certaines zones de texte sont inutiles ;
- d'autres étiquettes doivent être renommées et alignées verticalement.

C'est pourquoi, il faut mener quelques corrections, afin d'aboutir à la mise en page de la figure 87.

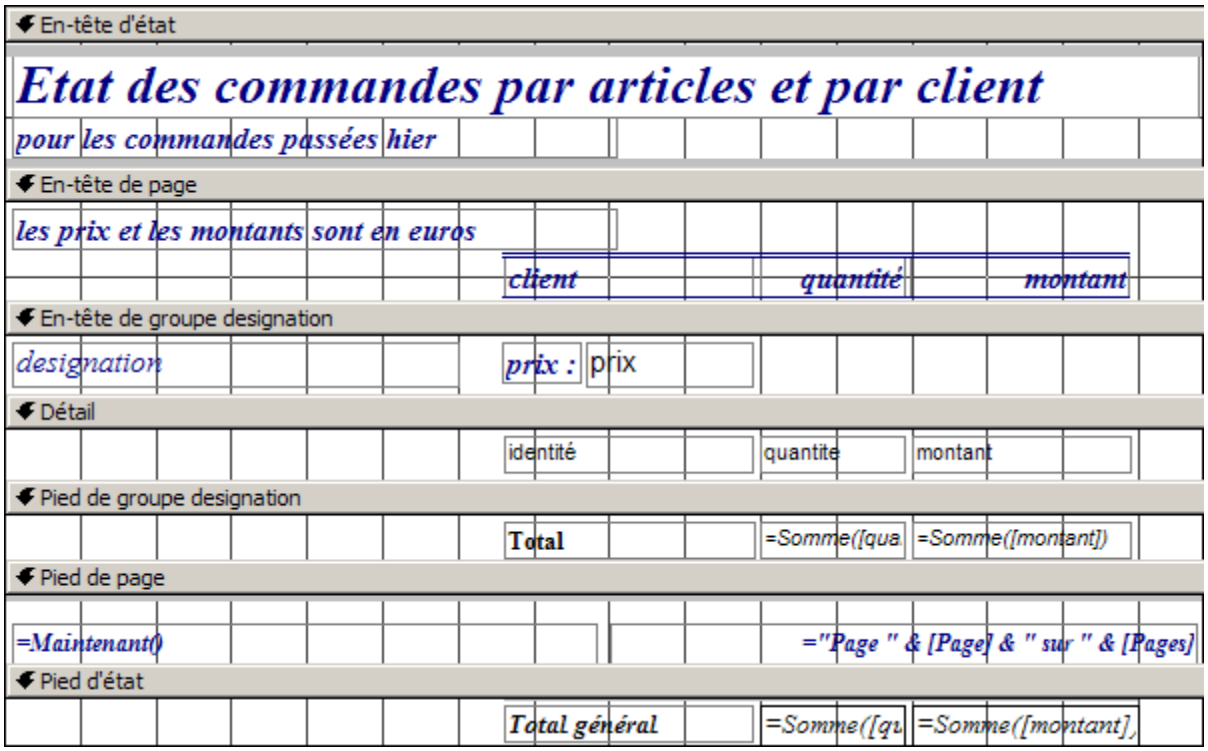


FIG. 87 – *Le même état, mais avec une disposition améliorée*

Cet état convient davantage car :
- les contrôles sont suffisamment larges pour les données actuelles[35] ;
- les conditions de sélection sont clairement affichées ;
- les unités également ;
- les intitulés du détail sont à leur place dans l'entête de page ;
- les étiquettes sont explicites et correctement organisées.

35. mais nous ne sommes jamais à l'abri d'un contenu trop long

Lorsque nous passons en mode affichage, nous obtenons le résultat attendu (figure 88).

Etat des commandes par articles et par client

pour les commandes passées hier

les prix et les montants sont en euros

	client	quantité	montant
50 DVD vierges	prix : 25		
	Albert Dubois	2	50
	André Dupuis	4	100
	Total	6	150
chargeur 4 AAA	prix : 15		
	Albert Dubois	1	15
	Arthur Dupont	2	30
	André Dupuis	1	15
	Total	4	60
clé USB 4 Go	prix : 150		
	Arthur Dupont	1	150
	André Dupuis	1	150
	Total	2	300
	Total général	12	510

FIG. 88 – *L'état corrigé en mode affichage est prêt à être imprimé et diffusé*

Cet état est, finalement, assez proche de l'état schématique de la figure 84 page 84.

6.1.5 Gestion des groupes et des tris

Pour modifier les niveaux de groupement et les tris d'un état après sa création par l'assistant, le bouton , disponible en mode création, permet d'ouvrir l'outil de la figure 89.

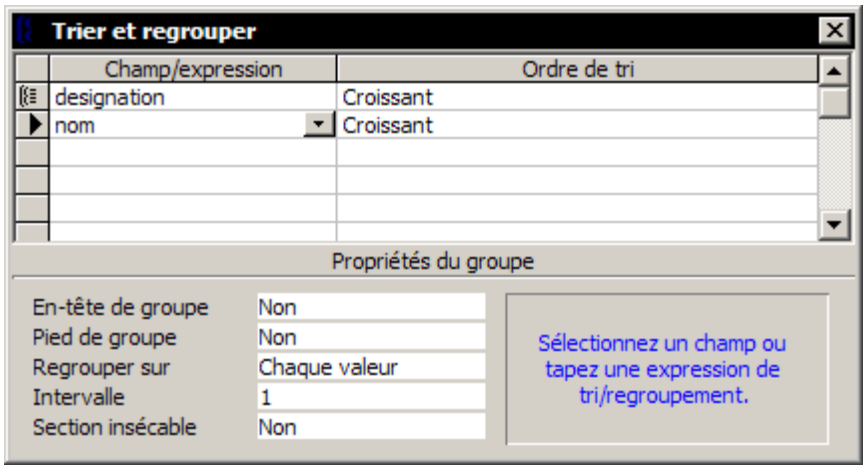


FIG. 89 – *Formulaire de modification des groupes et du sens des tris dans un formulaire*

Apparaissent alors les niveaux de groupement (hormis le niveau du détail) et les autres champs triés. Les niveaux de groupement (hormis le niveau du détail) sont automatiquement triés par Access, dans l'ordre croissant. Pour obtenir l'ordre décroissant, c'est dans cet outil qu'il faut intervenir.

Enfin, notons que la rangée `nom` n'est pas un niveau de groupement car elle ne possède ni en-tête de groupe, ni pied de groupe. Elle n'est donc pas affublée du symbole .

6.2 Annuaire

Pour un annuaire, nous retrouvons les mêmes éléments (figure 90) que l'état précédent.

	Annuaire des employés			← en-tête d'état
	identité	adresse	téléphone	← en-tête de page
A				← en-tête de groupe : initiale du nom
	Paul Acortia	18 rue ...	04 93 ...	← zone détail
	Marie Andrie	7 boulevard ...	03 27 ...	
	...			
B				
	Arthur Boulet	64 avenue ...	02 41 ...	
	...			
	page 1 / ...			← pied de page

FIG. 90 – *Composition d'un annuaire (échantillon)*

Les données nécessaires à cet annuaire se trouvent dans les tables dont le schéma relationnel est celui de la figure 91.

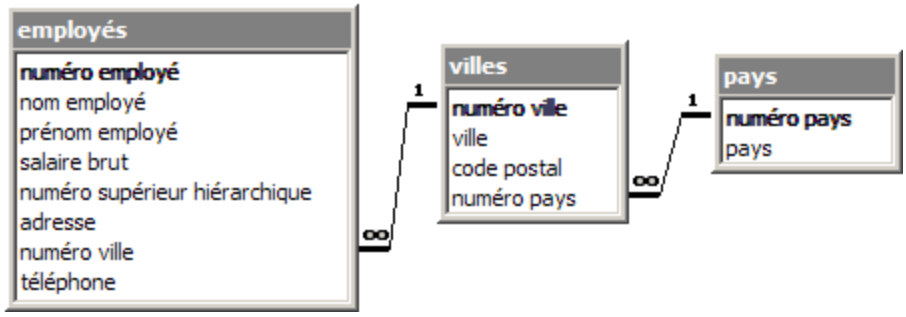
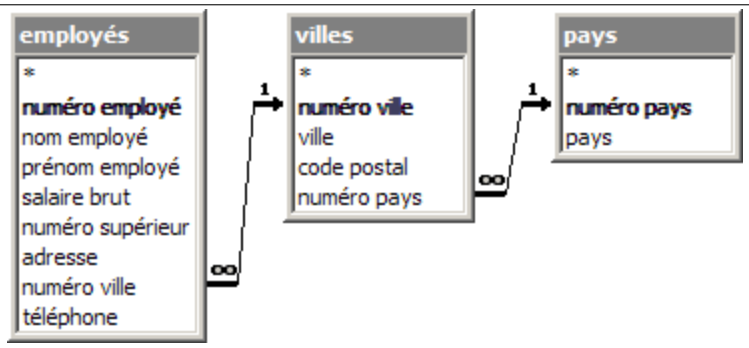


FIG. 91 – *Tables utilisées pour établir l'annuaire des employés*

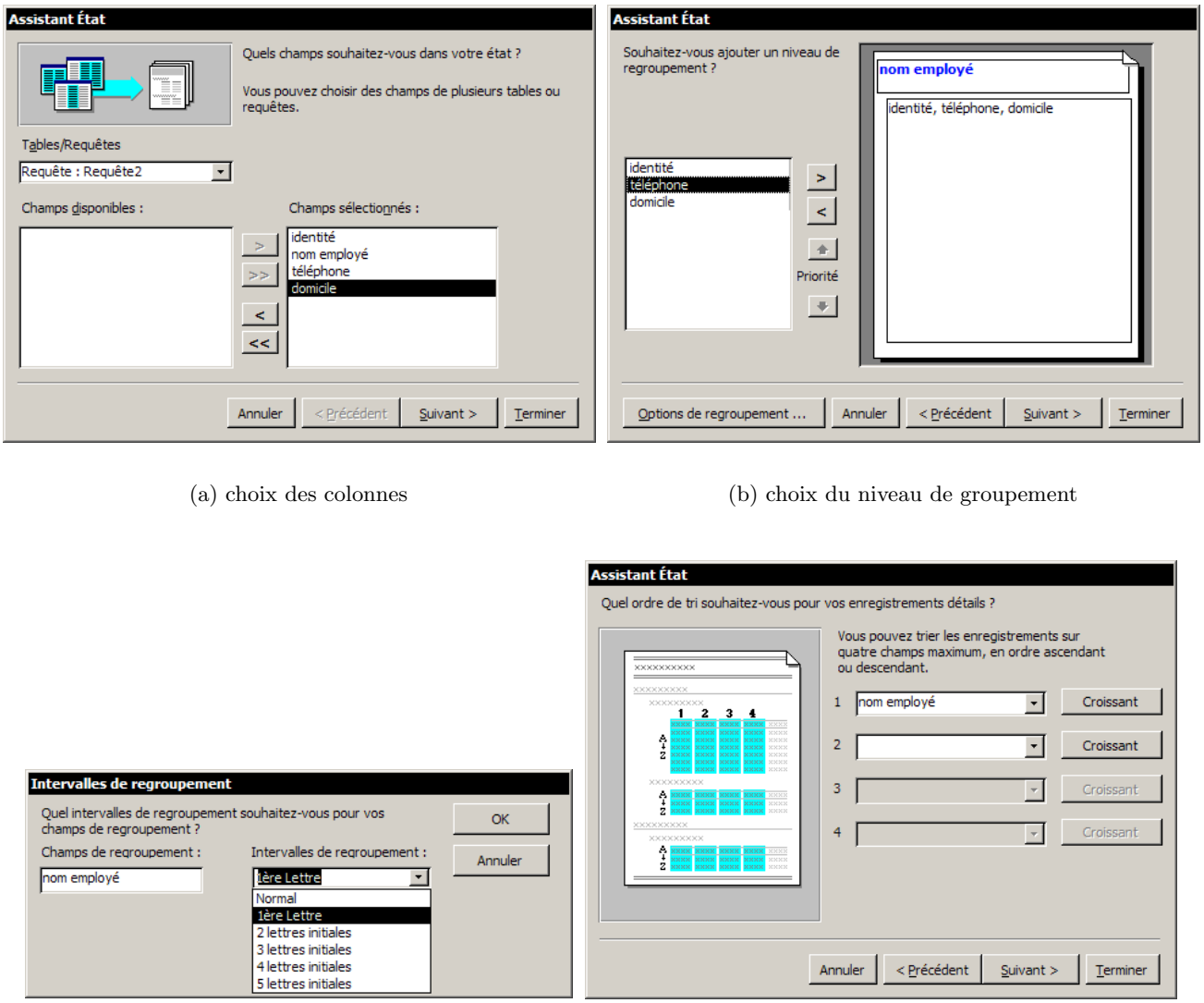
Le travail de transformation nécessaire (pour l'identité et l'adresse, notamment) est réalisé directement dans la requête de l'état (tableau 51).

les tables et leurs liaisons	
les colonnes et leur affichage	`identité: [prénom employé] & ' ' & [nom employé], [nom employé], téléphone, domicile: Nz(adresse;"") & ' ' & Nz([code postal];"") & ' ' & Nz(ville;"") & ' ' & Nz(pays;"")`
les critères sur les lignes	aucun
les tris et leur sens	`nom employé` (croissant)

TAB. 51 – *Requête pour un annuaire*

Pour obtenir un affichage de tous les employés, y compris pour les personnes dont nous ignorons le code postal, nous rendons facultatives les deux jointures de la requête (comme dans la section 3.9 page 45) et nous employons la fonction de dénullification `Nz`.

L'utilisation de l'assistant est similaire à celle de l'état de la section précédente, hormis l'absence d'agrégation et le choix de la première lettre du `nom employé` pour établir les intervalles du seul groupement (figure 92(c)).



(a) choix des colonnes

(b) choix du niveau de groupement

(c) choix des intervalles

(d) choix du tri

FIG. 92 – *Création d'un annuaire avec l'assistant*

La fenêtre de la figure 92(c) est obtenue en cliquant sur le bouton `Options de regroupement...` sur la figure 92(b).

Nous obtenons alors, en mode création, l'état de la figure 93.

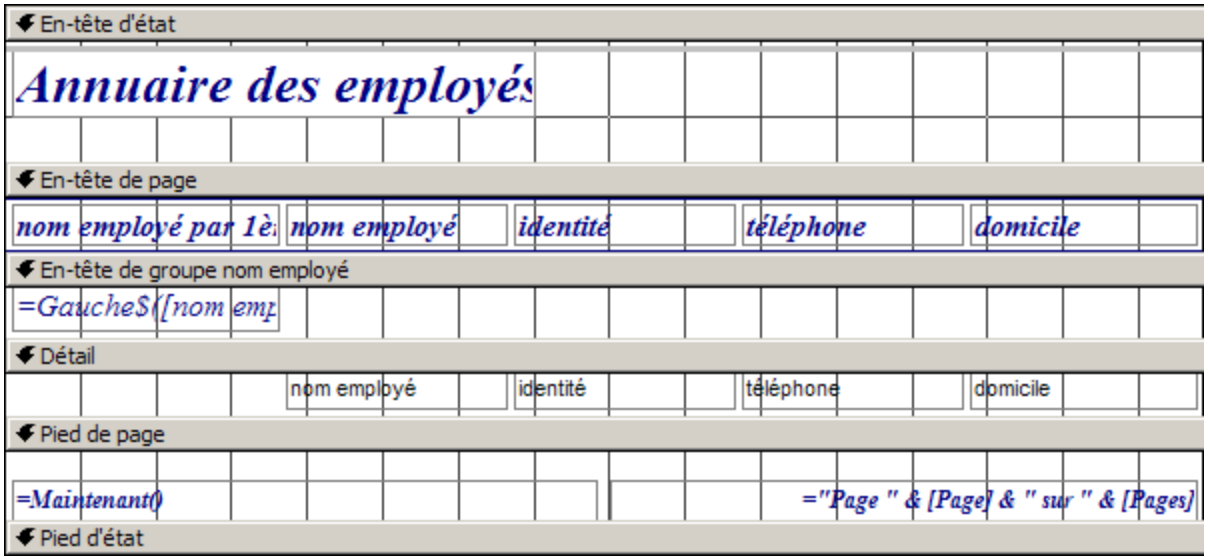


FIG. 93 – *Annuaire généré par l'assistant avant correction*

Cet état ne convient pas parfaitement mais moyennant quelques corrections[36] nous pouvons aboutir à la mise en page de la figure 94.

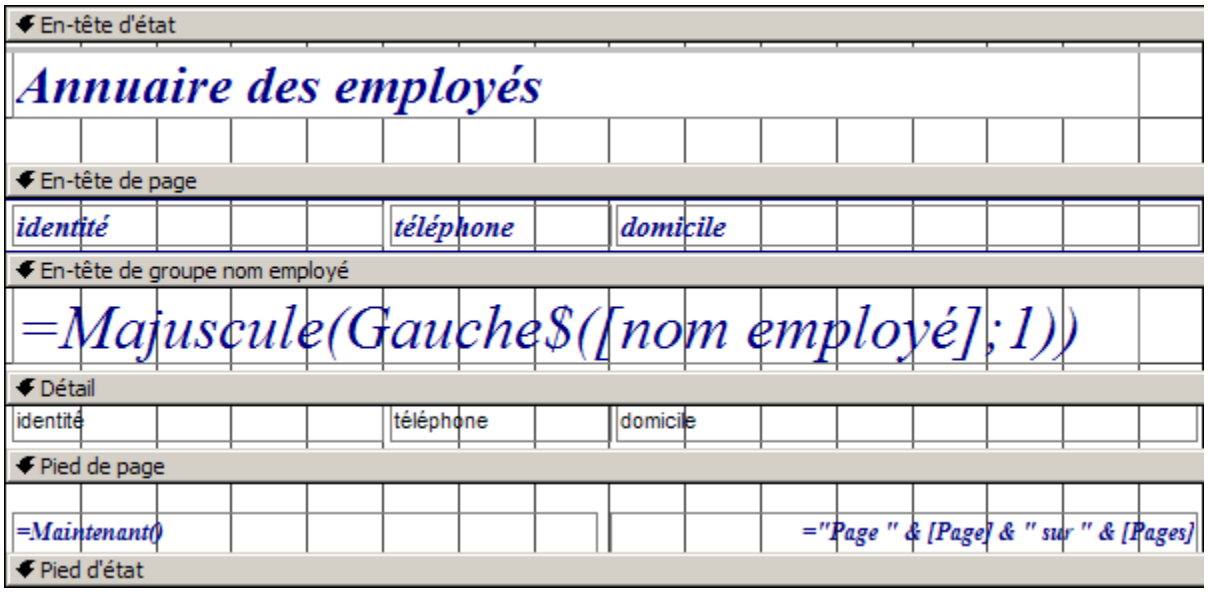


FIG. 94 – *Le même annuaire, mais mieux organisé*

36. dont l'ajout de la fonction `Majuscule` devant la fonction `Gauche`, afin de s'assurer que la lettre initiale sera bien majuscule (ce n'est pas le cas avec des personnes dont le nom commence par `von ...`, `van ...` ou `di ...`)

Lorsque nous passons en mode affichage, nous obtenons le résultat attendu (figure 95).

FIG. 95 – *L'annuaire corrigé en mode affichage est prêt à être imprimé et diffusé*

De nouveau, cet annuaire est assez proche de l'état schématique dessiné sur la figure 90 (page 90).

6.3 Mise en forme conditionnelle

Dans un état (mais aussi dans un formulaire), une zone de texte peut être mise en exergue, à la manière d'une cellule dans une feuille de calcul Excel, lorsque leur valeur passe un certain seuil, *via* la mise en forme conditionnelle. La mise en forme conditionnelle disponible dans le menu `Format`, offre la possibilité (assez rudimentaire et limitée à trois conditions, certes[37]) de changer la police, la taille et la couleur des caractères (figure 96).

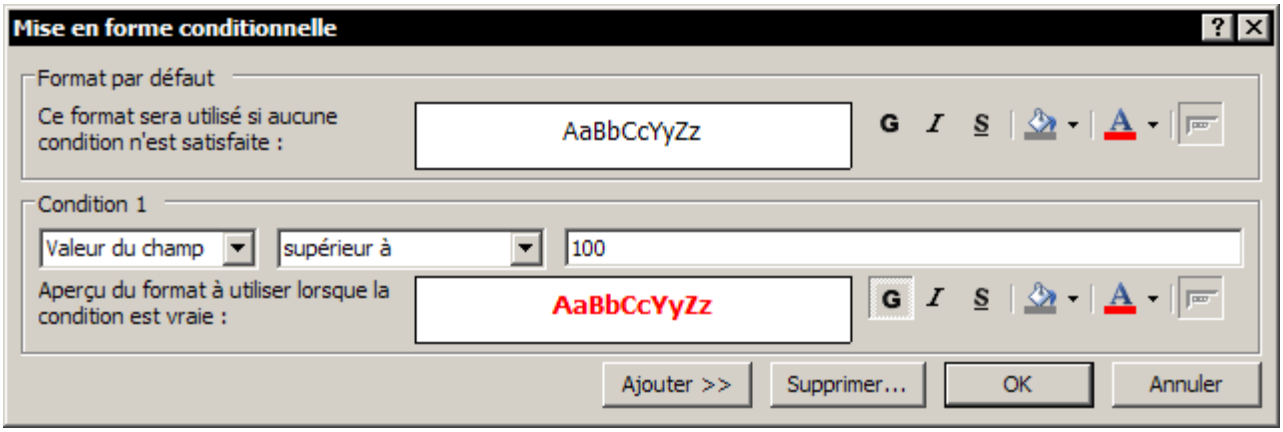


FIG. 96 – *Formulaire intégré pour une mise en forme conditionnelle*

L'exemple proposé sur cette figure est peu intéressant puisqu'il se contente d'afficher en rouge gras les prix supérieurs à 100 euros (c'est la zone de texte `prix` qui était active au moment ou la fenêtre `Mise en forme conditionnelle` a été ouverte).

37. un véritable outil d'édition d'états tel que Crystal Reports offre beaucoup plus de possibilité, tout en permettant de laisser les données dans un fichier Access

Mais à la place de la valeur `100`, nous aurions pu placer n'importe quelle formule, impliquant éventuellement l'intitulé d'un autre contrôle (ou d'une colonne non affichée[38]). Sur la figure 97, la mise en forme conditionnelle ne repose pas sur la valeur de la zone texte, mais sur une expression impliquant un booléen contenu dans une autre colonne de la requête de l'état.

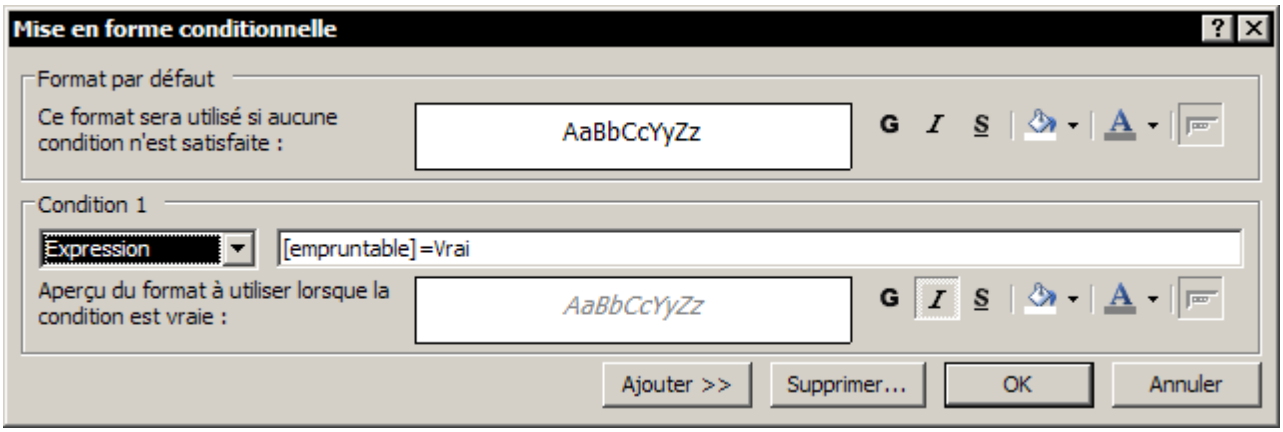


FIG. 97 – *Mise en forme conditionnelle basée sur une expression*

En revanche, pour afficher ou masquer un message selon qu'une expression est vraie ou fausse, ce n'est pas la mise en forme conditionnelle qui offre une solution. La solution se situe dans l'écriture de la requête de l'état, *via* l'utilisation de la fonction intégrée `VraiFaux` qui consiste à convertir la véracité d'une expression en une valeur si vrai et une autre valeur si faux.

Par exemple, pour afficher le message `livre non empruntable`, uniquement pour les livres non empruntables, il suffit d'ajouter à la requête de l'état la colonne :

```
nouvelle colonne: VraiFaux([empruntable];"";"livre non empruntable")
```

38. cela permet, par exemple, d'afficher en rouge gras les quantités inférieures à celle de l'année précédente, à la même date

6.4 Sous-états

Ordinairement, la zone détail sert à afficher une ligne de contrôles plusieurs fois, autrement dit un bloc d'informations plurielles. Pour afficher deux blocs d'informations plurielles côté à côte, il est nécessaire d'utiliser deux sous-états.

Par exemple, sur la figure 98, pour chaque livre du catalogue, nous souhaitons à la fois afficher les auteurs et les catégories, qui sont deux informations plurielles. Nous mettons donc, dans la zone détail, deux sous-états côte à côte, à l'aide du bouton .

Catalogue des livres sur SQL	← en-tête d'état
titre : SQL3 : initiation, programmation	← en-tête de groupe
auteur(s) / catégorie(s)	← zone détail : présence de deux sous-états
Christian Robins / informatique	
Guy Guarno / guide pratique	
Didier Grint	
titre : SQL Server 2005	
auteur(s) / catégorie(s)	
Marc Darson / informatique	
manuel de formation	
page 1 / ...	← pied de page

FIG. 98 – *Utilisation de deux sous-états*

Le schéma relationnel des tables impliquées dans l'état précédent est celui de la figure 99. Nous y voyons la présence de deux liaisons plusieurs-à-plusieurs qui sont à l'origine des deux sous-états.

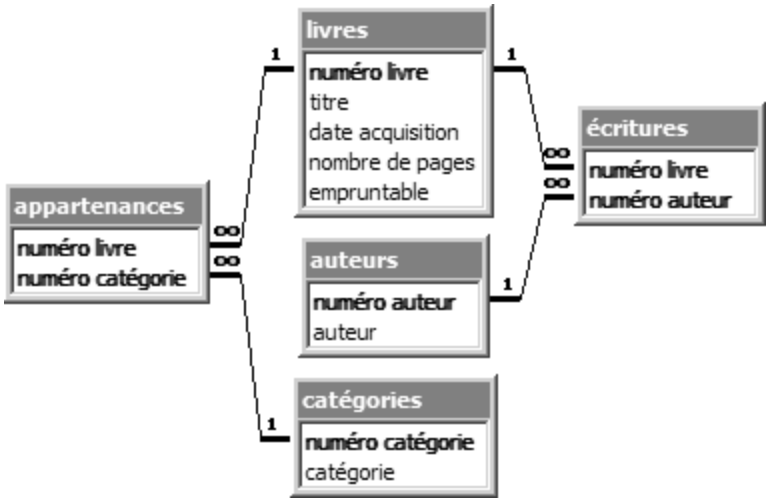


FIG. 99 – *Tables utilisées pour l'état comportant deux sous-états*

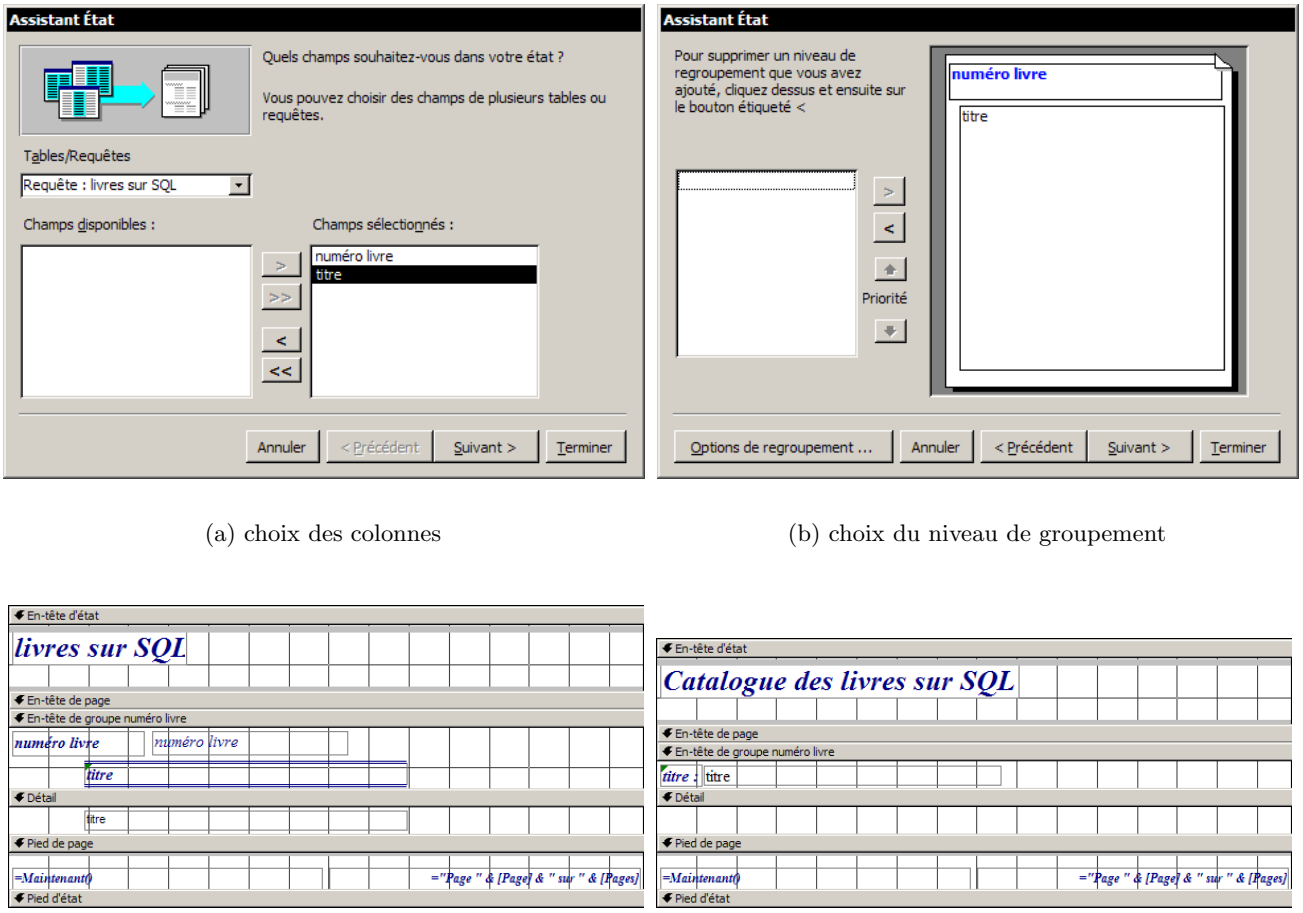
Si, sur cet exemple, l'état comporte deux sous-états, c'est parce qu'un état qui n'utiliserait qu'un seul sous-état pourrait être remplacé par un état sans sous-état. Le principe-même d'un état est de fonctionner comme un sous-état imbriqué dans un sous-état, lui-même imbriqué dans un sous-état, etc., autant de fois qu'il y a de niveau de regroupement.

6.4.1 Création de l'état principal

Si nous n'avions que la liste des auteurs à afficher, la zone détail de l'état principal serait répétée autant de fois qu'il y a d'auteurs pour un livre. Donc si nous ajoutons dans cette zone détail un sous-état pour afficher les catégories, alors toutes les catégories d'un livre seront répétées autant de fois qu'il y a d'auteurs, ce qui ne correspond pas à ce que nous souhaitons.

Pour éviter que la zone détail ne soit répétée plusieurs fois pour un même livre, nous créons donc, dans une première étape, un état principal dont la zone détail est répétée une seule fois. Ce n'est que plus tard que nous y ajouterons les deux sous-états.

Cet état principal est très simple, il se contente d'afficher les informations issues dans la table des livres (figure 100).



(a) choix des colonnes

(b) choix du niveau de groupement

(c) résultat

(d) correction

FIG. 100 – *Création de l'état principal avec l'assistant*

La requête `livres sur SQL` de la figure 100(a) sélectionne simplement les colonnes `numéro livre` et `titre` de la table `livres`, avec `titre Comme "*SQL*"` pour critère sur les lignes. Pour que la zone détail

soit répétée au moins une fois, il nous faut un niveau de groupement (figure 100(a)) et un seul, quitte à supprimer l'affichage du `numéro livre` et à déplacer le `titre` dans l'entête (figures 100(c)).

La zone détail, vierge pour le moment, est prête à recevoir ses deux sous-états.

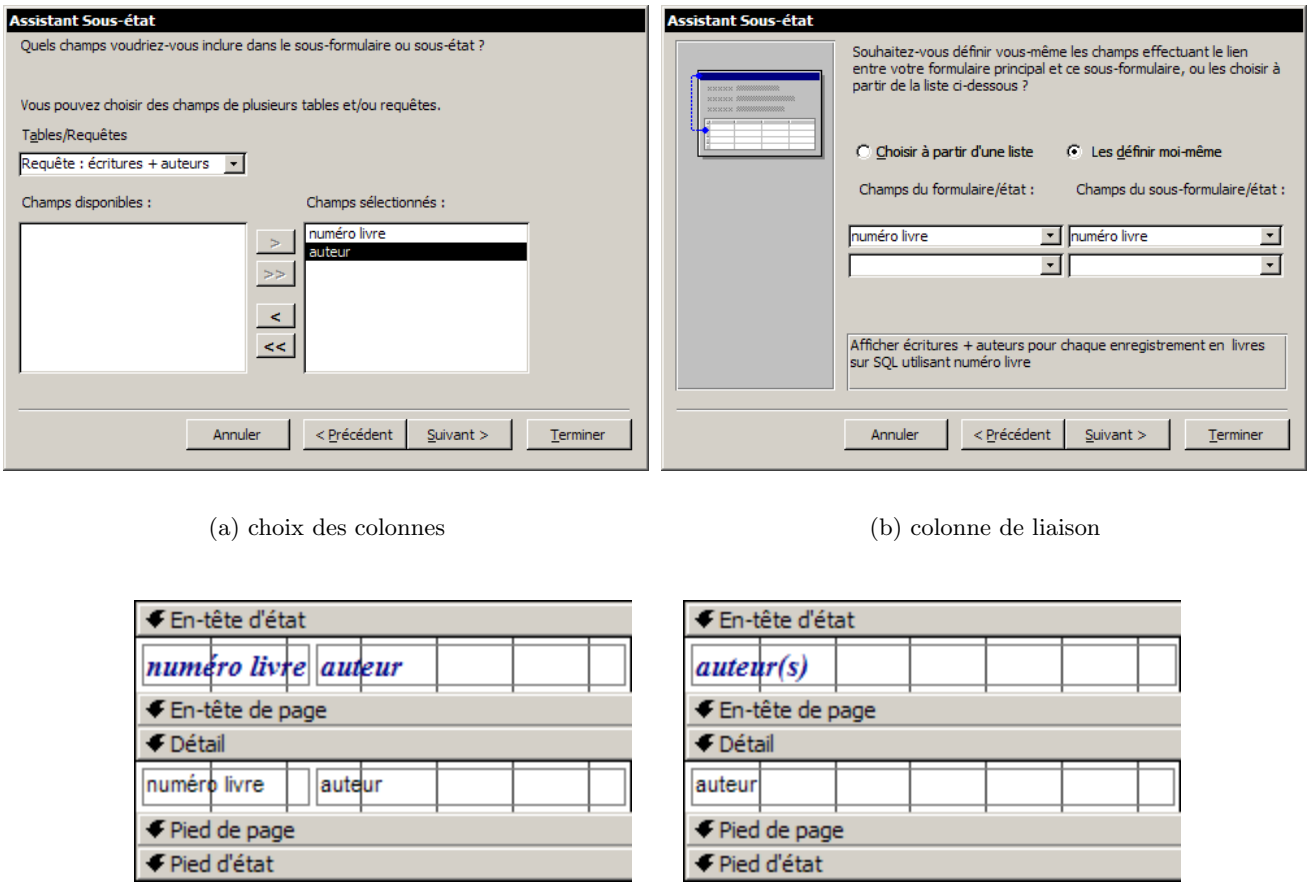
6.4.2 Création des deux sous-états

Le sous-état des auteurs repose sur la requête `écritures + auteurs` du tableau 52, qu'il faut écrire au préalable.

les tables et leurs liaisons	écritures, auteurs (ces tables sont correctement liées sur le schéma relationnel)
les colonnes et leur affichage	`numéro livre`, `auteur`
les critères sur les lignes	aucun
les tris et leur sens	aucun

TAB. 52 – *Requête nommée* `écritures + auteurs` *pour le premier sous-état*

Ensuite, nous ajoutons un sous-état à la zone détail de l'état principal, en prenant soin de vérifier que le bouton magique est bien enclenché pour bénéficier de l'assistant (figure 101(a)).



(a) choix des colonnes

(b) colonne de liaison

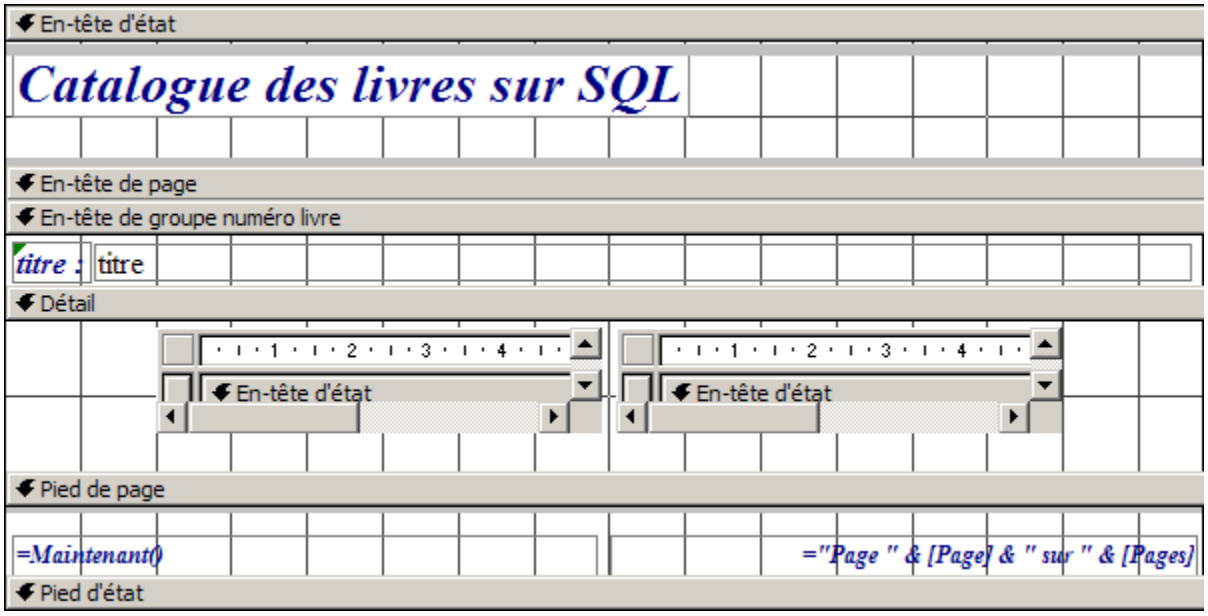
(c) résultat

(d) correction

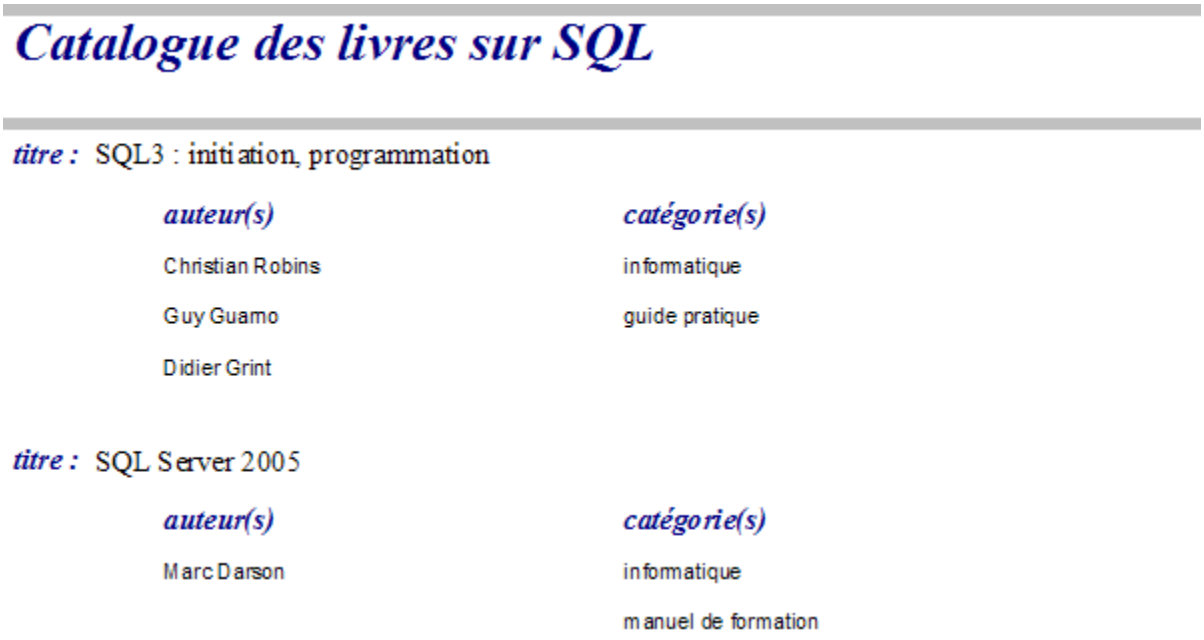
FIG. 101 – *Création du premier sous-état avec l'assistant*

La liaison du sous-état avec l'état principal se fait sur la colonne commune `numéro livre` (figure 101(b)). Nous obtenons le sous-état de la figure 101(c), que nous avons ouvert dans une nouvelle fenêtre à l'aide du menu `Affichage`, `Sous-état dans une nouvelle fenêtre`, après l'avoir sélectionné. Nous pouvons corriger le résultat obtenu afin de masquer le `numéro livre` (figure 101(d)).

Il reste à faire de même pour les catégories et nous obtenons l'état principal complet de la figure 102(a).



(a) en mode création



(b) en mode aperçu

FIG. 102 – *L'état principal complet, muni de ses deux sous-états dans la zone détail*

Lorsque nous passons en mode affichage, nous obtenons le résultat attendu (figure 102(b)).

7 Visual Basic for Applications

À plusieurs reprises dans la section dédiée aux formulaires, il est apparu que l'ergonomie serait améliorée si nous pouvions programmer certaines actions, en réponse à des événements. Pour tous les traitements spécifiques non prévus par Access, le développeur d'un formulaire (ou d'un état) peut écrire un code adéquat en VBA. L'accès au code se fait généralement en cliquant sur le bouton qui fait apparaître la fenêtre Visual Basic for Applications (figure 103).

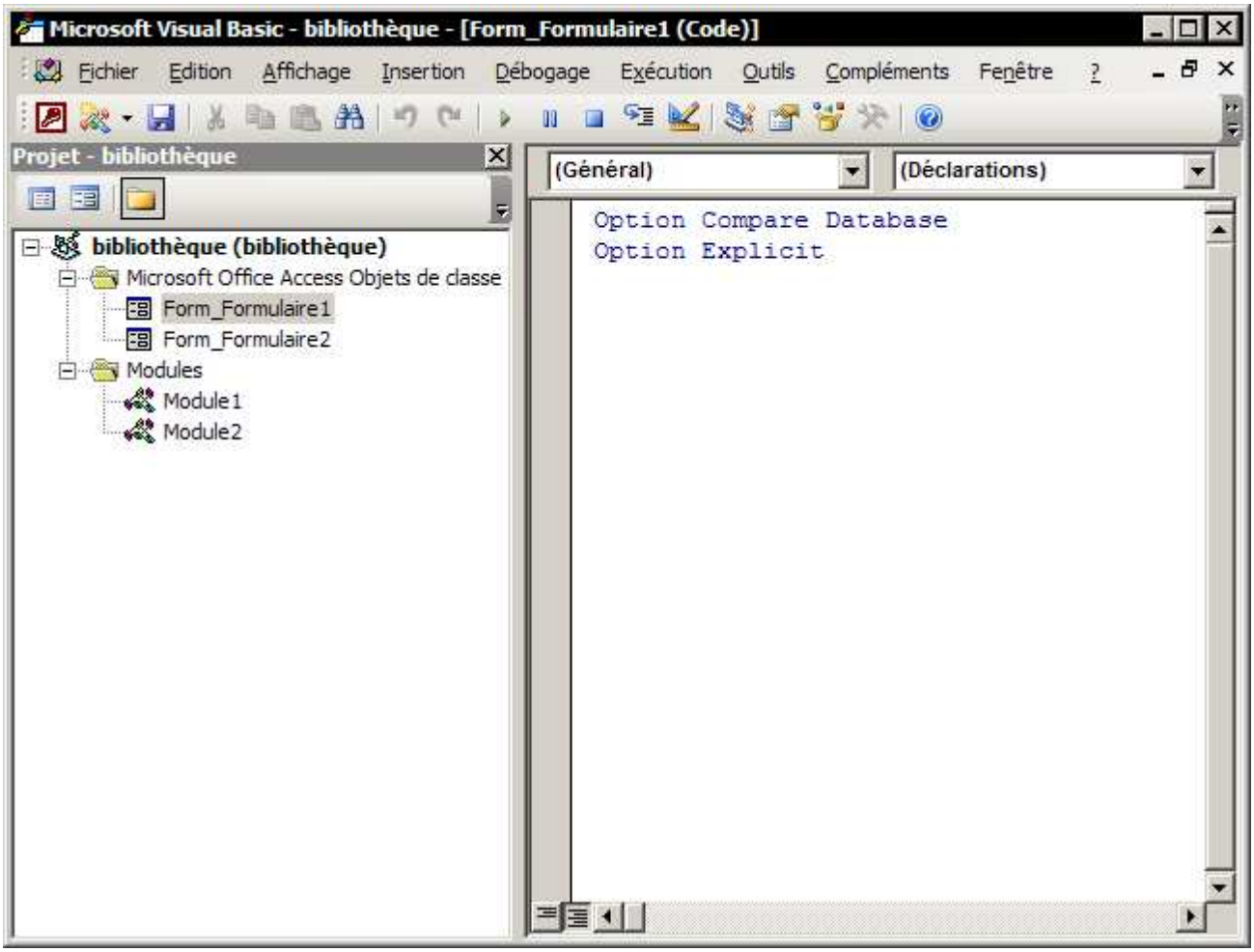


FIG. 103 – *Fenêtre Visual Basic for Applications avec, à gauche, la liste des modules et à droite, la feuille de programmation du module sélectionné*

Chaque formulaire et chaque état dispose d'un module VBA, au début duquel figurent les deux options `Compare` et `Explicit` qui signifient respectivement, que la comparaison des chaînes de caractères tiendra compte (ou non) des majuscules / minuscules et que toutes les variables utilisées devront être déclarées au préalable (ce qui est très utile).

Une fois le travail terminé dans la fenêtre Visual Basic for Applications, le retour à Access peut se faire avec le bouton .

7.1 Description du langage de programmation

Comme pour tout langage de programmation, commençons par énoncer les bonnes manières à adopter dans le code. Tout d'abord, nous pouvons mettre plusieurs instructions sur une ligne à l'aide des deux points `:`. Mais, pour des raisons de lisibilité, il est conseillé de ne pas mettre plus d'une instruction par ligne.

De plus, nous pouvons étaler une longue instruction sur plusieurs lignes avec le tiret bas _ précédé d'une espace[39]. Nous en profitons donc pour couper les lignes qui dépassent de l'écran et pour indenter comme dans ce document[40].

Par ailleurs, un programme bien commenté est plus facile à relire et à corriger. Nous utilisons donc l'apostrophe `'` chaque fois que cela est nécessaire, car tout ce qui se trouve après, dans le code VBA, est considéré comme un commentaire.

Enfin, il est conseillé d'utiliser la complétion automatique (en appuyant sur `CTRL + espace` après avoir tapé les premières lettres d'un mot) afin d'éviter au maximum les erreurs de syntaxe. Cela permet d'employer des noms de variables suffisamment longs pour en décrire le contenu, sans avoir à souffrir de leur longueur, à la saisie.

7.1.1 Types de données

En VBA, comme dans tout langage informatique, chaque constante et chaque variable possède un type de données. Ces types de données (tableau 53) sont légèrement différents de ceux des colonnes d'Access (tableau 1 page 7).

type	description	taille[a]	description
`Boolean`	`True` ou `False`	2 octets	gaspillage car 1 bit suffirait
`Byte`	un caractère	1 octet	vaut entre 0 et 255
`String`	une chaîne de caractères	entre 1 et 65 535 caractères d'un octet chacun	
`Integer`	un entier	2 octets	entre -32 768 et 32 767
`Long`		4 octets	entre -2 147 483 648 et 2 147 483 647
`Single`	un réel à virgule flottante	4 octets	$\pm 3\,10^{38}$ avec 7 décimales exactes
`Double`		8 octets	$\pm 1.7\,10^{308}$ avec 15 décimales exactes
`Currency`	un réel à virgule fixe (4 décimales après la virgule)	8 octets	$\pm 9\,10^{15}$ (toutes décimales exactes)
`Date`	date <u>et</u> heure	8 octets	entre le 01/01/100 et le 31/12/9999 et entre 0:00:00 et 23:59:59

TAB. 53 – *Types de données natifs de Visual Basic for Applications*

[a] un octet représente 8 positions binaires

Tous les types de données utilisables par le développeur VBA ne figurent pas dans ce tableau. Il existe d'autres types, spécifiques à Access ou à Excel, comme `Worksheet`qui représente une feuille de calcul et qui est en réalité une référence, c'est-à-dire une adresse (codée sur 4 octets). De plus, le développeur peut définir de nouvelles classes d'objets qui sont autant de nouveaux types de données. Pourtant, la majorités des formulaires et des états ne nécessitent pas le développement de nouvelles classes d'objet *via* l'ajout d'un module avec le bouton . Nous n'abordons donc pas cette fonctionnalité d'Access dans ce document et nous laissons le lecteur consulter la référence [Tylor et Anderson] à ce sujet.

39. en typographie, espace est toujours un nom féminin
40. l'indentation est l'utilisation de tabulations dans un texte pour aligner verticalement les éléments d'un même niveau

7.1.2 Déclaration et affectation

Pour déclarer une constante, nous utilisons l'instruction `Const ... As ...` suivie de sa valeur :

```
Const NombreJoursDansSemaine As Integer = 7
Const NombreMoisDansAnnee As Integer = 12
Const NombreHeuresDansSemaine As Integer = NombreJoursDansSemaine * 24
```

Tandis que pour déclarer une variable, nous utilisons l'instruction `Dim ... As ...`, l'affectation (ou les affectations) se fera (se feront) plus loin :

```
Dim NumeroHeureDansSemaine As Integer
Dim NumeroJour, NumeroHeure, NumeroMois As Integer
Dim FeuilleActive As Worksheet
Dim Mois(1 To 12) As String ' tableau de 12 chaînes de caractères
                            ' dont le premier indice est 1
                            ' (au lieu de 0 par défaut)
Dim Jours (1 To NombreJoursDansSemaine) As String
Dim JoursParMois (1 To NombreMoisDansAnnee, 1 To 31) As String
  ' matrice de 12 lignes et de 31 colonnes
```

Le nom d'une constante ou d'une variable ne peut pas comporter d'accent et, rappelons-le, doit décrire ce qu'elle contient et non de quel type elle est (car cette information est disponible *via* le menu contextuel (clic droit) info express

). De plus, la différence entre les majuscules et les minuscules n'est pas faite par VBA (sauf pour le contenu des chaînes de caractères, heureusement), à telle enseigne que l'interpréteur de code corrige immédiatement un nom qui aurait une majuscule ou une minuscule différente du nom qui lui a été donné à la déclaration.

7.1.3 Opérateurs et affichage

Pour l'instant, les variables déclarées précédemment n'ont aucune valeur. Il reste donc à les affecter avec l'opérateur `=` (précédé ou non de l'instruction `Let`, facultative) :

```
Let NumeroHeureDansSemaine = 53
NumeroHeure = NumeroHeureDansSemaine Mod 24 ' opérateur modulo
NumeroJour = 1 + NumeroHeureDansSemaine / 24
Jours(1) = "lundi"
Jours(2) = "mardi"
' etc.
JoursParMois(1,1) = Jours(4)
```

Pour une référence, l'instruction `Let` est remplacée par une instruction `Set` qui, elle, est obligatoire :

```
Set FeuilleActive = Sheets("Feuil1")
  ' Sheets est la collection des feuilles du classeur
  ' Feuil1 est le nom de la feuille voulue
```

Pour concaténer des chaînes de caractères, l'opérateur est à nouveau l'esperluette `&`, et les chaînes constantes sont encadrées par des doubles quotes `"` (y compris si elles sont réduites à un caractère) :

```
Dim AujourdHui As String
AujourdHui = "Nous sommes le " & Jours(NumeroJour) & " " & Date()
  ' Date retourne la date courante
```

Pour afficher une chaîne de caractères ou le contenu de toute autre variable dans une boîte de dialogue, nous disposons de l'instruction `MsgBox` :

```
1 MsgBox(AujourdHui)
```

L'exécution de cette instruction produit l'affichage de la fenêtre de la figure 104.

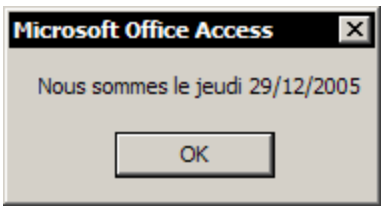


FIG. 104 – *Boîte de dialogue* `MsgBox`

Le lecteur est invité à interroger l'aide d'Access pour découvrir les nombreuses options disponibles pour l'instruction `MsgBox`.

7.1.4 Procédures et fonctions

Dans chaque module VBA nous pouvons programmer des procédures et des fonctions en les ajoutant successivement à l'aide du bouton qui fait apparaître la boîte de dialogue de la figure 105.

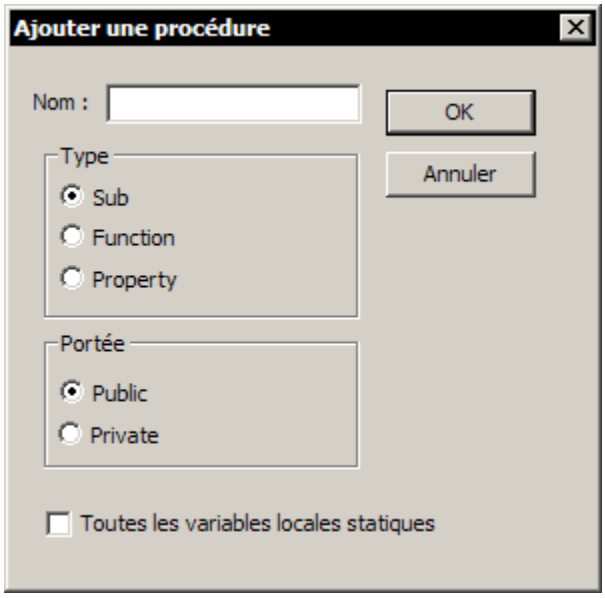


FIG. 105 – *Boîte de dialogue pour insérer une nouvelle routine dans le module*

La déclaration d'une telle routine se fait en indiquant sur la première ligne :

- la visibilité de la routine : `Public` pour une routine utilisable dans tous les modules ou `Private` pour une routine utilisable uniquement dans son module ;
- la nature de la routine : `Sub` pour une procédure (qui est une routine sans valeur de renvoi, mais avec des arguments d'entrée et de sortie[41]) ou `Function` pour une fonction (qui est une routine avec une valeur de renvoi et des arguments d'entrée) ;
- le nom de la routine : il obéit aux mêmes règles syntaxiques que les noms de variable et doit décrire ce que fait la routine ;
- les arguments et leurs types respectifs ;
- et pour les fonctions, le type de la valeur de renvoi.

Par exemple, considérons la procédure publique et la fonction publique suivantes :

```
Public Sub AfficherJour (NumeroJour As Integer, Jours() As String)
  MsgBox("Nous sommes le " & Jours(NumeroJour) & " " & Date)
End Sub

Public Function JourDansSemaine() As Integer
  JourDansSemaine = Datepart("w", Date)
  ' Datepart est une fonction VBA qui existe déjà dans Access
End Function
```

Lorsqu'un argument est un tableautableau ou une matrice, il est indispensable de rajouter, à son nom, une parenthèse ouvrante et une parenthèse fermante (comme ci-dessus pour le deuxième argument de la procédure). De même, si une routine n'a pas besoin d'argument[42], il faut malgré tout indiquer une parenthèse ouvrante et une parenthèse fermante pour le signaler (c'est le cas de notre fonction ci-dessus).

Cette procédure et cette fonction permettent de ne pas avoir à répéter leur contenu à chaque fois que nous avons besoin d'afficher la date ou de calculer le jour de la semaine. Il suffit de les appeler et de leur passer leur(s) argument(s) ainsi :

```
NumeroJour = JourDansSemaine() ' NumeroJour a déjà été déclaré
Call AfficherJour(NumeroJour, Jours)
```

L'appel à une fonction ne nécessite pas l'emploi de l'instruction `Call`, mais nécessite la déclaration préalable d'une variable dans laquelle son résultat sera stocké.

41. accompagnés éventuellement d'argument(s) qui serve(nt) d'espace de travail
42. elle est alors niladique

7.1.5 Structures de décision

Avec VBA, nous retrouvons les quatre structures ou instructions de décision qui font les beaux jours de l'algorithmique (tableau 54).

cas de figure	deux choix possibles	trois choix ou plus	
une valeur à choisir	`IIf`(condition, valeur si vrai, valeur si faux)	`Choose`(index, valeur1, valeur2, valeur3, ...)	

blocs d'instructions à choisir	`If` condition `Then`
	instructions
	`Else`
	instructions
	`End If`
		`Select Case` variable
		`Case` valeur1
		instructions
		`Case` valeur2
		instructions
		...
		`Case Else`
		instructions
		`End Select`
			`If` condition1 `Then`
			instructions
			`Else If` condition2
			instructions
			...
			`Else`
			instructions
			`End If`

Tab. 54 – *Branchements conditionnels*

L'instruction `IIf` est très pratique, car elle peut être utilisée dans une formule :

```
Dim Semestre As String
Semestre = IIf(NumeroMois <= 6, "premier", "deuxième")
```

La structure `If ... Then` (pour laquelle le `Else` est facultatif), est très utile pour gérer la réponse d'une `MsgBox` de type « oui ou non » :

```
Dim Reponse As Integer
Reponse = MsgBox("Voulez-vous continuer ?", vbYesNo)
'vbYesNo ajoute deux boutons Oui et Non

If Reponse = vbNo Then ' vbNo est une constante VBA
  GoTo Fin             ' branchement inconditionnel à l'étiquette Fin
End If

...                    ' instructions pour continuer

Fin:                   ' étiquette Fin (placée où nous voulons, en fait)

...                    ' instructions pour terminer
```

La condition utilisée dans la structure `If ... Then` peut être aussi complexe que nécessaire et peut faire appel aux opérateurs logiques[43] `And`, `Or` et `Not`, ainsi qu'à tous les opérateurs de comparaison classiques :

```
Dim A, B, C, D As Integer
Dim E As String
... ' affectation de A, B, C, D et E

If Not((A < B And C >= D) Or E <> Empty) Then
  ...
Else
  ...
End If
```

`Empty` est une constante VBA qui représente la chaîne vide `""`.

L'instruction `Choose` admet quant à elle autant d'arguments que nécessaire pour faire son choix :

```
Dim Mois As String
Mois = Choose(NumeroMois, "janvier", "février", "mars", "avril", "mai", "juin", _
                          "juillet", "août", "septembre", "octobre", "novembre", _
                          "décembre")
```

La structure `Select Case` une seule variable peut être testée (s'il faut tester plusieurs variables à la fois, c'est alors la structure `If ... Then ... Else If` qu'il faut employer) :

```
Public Function NombreJours(NumeroMois As Integer, Annee As Integer) As Integer
  Select Case NumeroMois
    Case 1, 3, 5, 7, 8, 10, 12
      NombreJours = 31
    Case 2
      NombreJours = IIf((Annee Mod 4 = 0) And _
                        (Annee Mod 100 <> 0 Or Annee Mod 400 = 0), 29, 28)
    Case Else
      NombreJours = 30
    End Select
End Function
```

Nous voyons sur cet exemple, que les structures de décision peuvent être imbriquées les unes dans les autres.

Il est également permis d'utiliser `Case 1 To 12` et `Case Is > 12` (autrement dit, une plage de valeurs consécutives ou une comparaison).

43. à parenthéser correctement, en cas de doutes sur les priorités

7.1.6 Structures de répétition

De manière tout aussi classique, VBA offre les deux types de boucles : les boucles `For` dont la répétition est contrôlée par un indice et les boucles `While` qui peuvent ne jamais s'arrêter (tableau 55).

cas de figure	nombre de répétitions inconnu	nombre connu à l'avance
la boucle peut ne pas être exécutée	`Do While` condition instructions `Loop`	`For` indice = début `To` fin instructions `Next` indice
au moins une exécution	`Do` instructions `Loop Until` condition	s'assurer que début < fin

TAB. 55 – *Boucles répétitives*

Une boucle `For` permet, par exemple, de calculer la moyenne d'un tableau de valeurs :

```
Public Function Moyenne(Valeurs() As Double) As Double
  Dim NumeroValeur As Integer
  Dim Somme As Double

  For NumeroValeur = LBound(Valeurs()) To UBound(Valeurs())
  ' LBound et UBound donnent les bornes des indices du tableau
    Somme = Somme + Valeurs(NumeroValeur)
  Next NumeroValeur ' incrémentation automatique de l'indice

  Moyenne = Somme / (UBound(Valeurs()) - LBound(Valeurs())
  ' heureusement, Somme est un Double, la division ne sera donc pas entière
End Function
```

La structure `For` autorise également une syntaxe du type `For i = 0 To 121 Step 7`, pour que l'indice s'incrémente de 7 en 7.

Une boucle `While` permet, par exemple, de concaténer des chaînes de caractères jusqu'à tomber sur une chaîne vide :

```
Public Function Concatenation(Chaines() As String) As String
  Dim NumeroChaine As Integer
  Dim Resultat As String

  NumeroChaine = LBound(Chaines())

  Do While Chaines(NumeroChaine) <> Empty
    Resultat = Resultat & Chaines(NumeroChaine)
    NumeroChaine = NumeroChaine + 1 ' sinon, la boucle stagne au même indice
  Loop

  Concatenation = Resultat
End Function
```

Le plus important avec une boucle `While` est d'ajouter un indice de boucle et de ne pas oublier de faire soi-même l'incrémentation de cet indice, sans quoi la boucle pourrait tourner indéfiniment.

Par ailleurs, il est parfois nécessaire de sortir prématurément d'une boucle. Pour sortir d'une boucle

`For` avant la fin, nous pouvons utiliser l'instruction `Exit For` :

```
Public Function Chercher(Chaine As String, Chaines() As String) As Integer
  Dim NumeroChaine As Integer
  Dim Trouve As Boolean

  Trouve = False ' au départ, rien n'est trouvé

  For NumeroChaine = LBound(Chaines()) To UBound(Chaines())
    If Chaines(NumeroChaine) = Chaine Then
      Trouve = True
      Exit For ' passe à la première instruction après la boucle
    End If
  Next NumeroChaine

  Chercher = IIf(Trouve = True, NumeroChaine,  LBound(Chaines()) - 1)
  ' si la chaîne n'est pas trouvée, alors le code appelant la fonction Chercher
  ' recevra LBound(Chaines()) - 1 et devra gérer cette valeur (c'est un choix)
End Sub
```

Nous voyons sur cet exemple, que les boucles et les structures de décision peuvent parfaitement être imbriquées les unes dans les autres.

Pour sortir d'une boucle `While` ou `Until`, nous disposons de l'instruction `Exit Do`. Malheureusement, `Exit For` et `Exit Do` ne permettent de sortir que d'une boucle à la fois. Pour sortir de plusieurs boucles à la fois, le plus simple et d'utiliser l'instruction `GoTo` avec une étiquette à l'endroit désiré :

```
Public Sub ChercherMatrice(Chaine As String, Chaines() As String, _
                           Ligne As Integer, Colonne As Integer)
' cette fois, il s'agit d'une procédure
' car deux valeurs doivent être retournées en sortie
  Dim NumeroLigne, NumeroColonne As Integer
  Dim Trouve As Boolean

  Trouve = False

  For NumeroLigne = LBound(Chaines(), 1) To UBound(Chaines(), 1)
    For NumeroColonne = LBound(Chaines(), 2) To UBound(Chaines(), 2)
      If Chaines(NumeroLigne, NumeroColonne) = Chaine Then
        Trouve = True
        GoTo Fin ' passe à l'étiquette Fin
      End If
    Next NumeroColonne
  Next NumeroLigne

Fin:
  Ligne = IIf(Trouve = True, NumeroLigne,  LBound(Chaines(), 1) - 1)
  Colonne = IIf(Trouve = True, NumeroColonne,  LBound(Chaines(), 2) - 1)
End Sub
```

Nous voyons sur cet exemple, l'imbrication de deux boucles `For` pour la recherche dans une matrice et nous voyons également l'utilité du deuxième argument des fonctions `LBound` et `UBound`.

7.2 Objets et événements

VBA est un langage orienté objet, accompagné d'une bibliothèque dont une partie est disponible pour Access (pour Excel également). Le développeur peut naviguer au sein de cette bibliothèque en faisant apparaître l'explorateur d'objets (figure 106) disponible grâce au bouton .

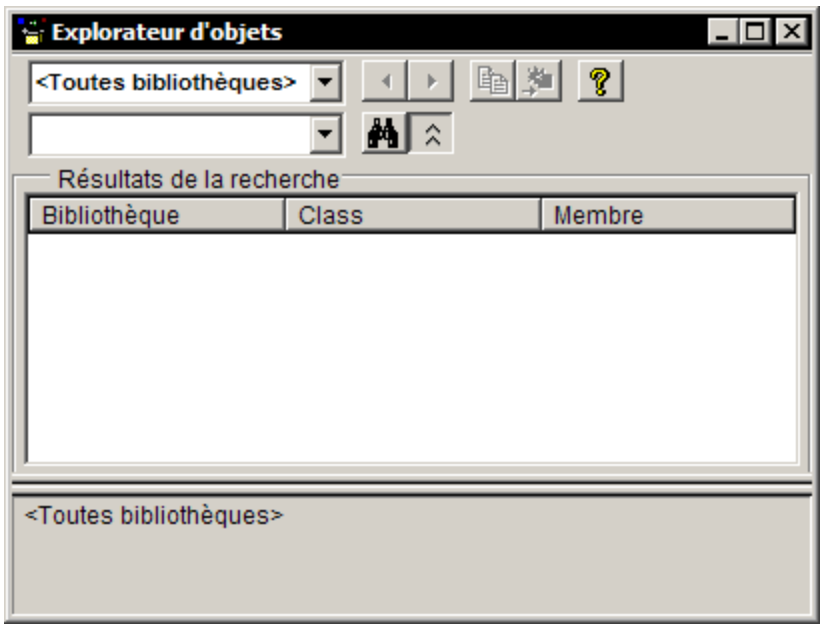


FIG. 106 – *Explorateur d'objets VBA*

Les objets les plus utilisés dans Access sont : les formulaires, les états et les contrôles (dont les boutons). Les membres des objets, c'est-à-dire leurs propriétés et les méthodes (ou fonctions) que nous pouvons leur appliquer, sont définis dans les classes d'objets. Sont également définis dans ces classes, les événements auxquels répondent les objets d'une même classe.

7.2.1 Contrôles

En ce qui concerne les contrôles, le nom des principales classes sont listées dans le tableau 56, ainsi que leurs principales propriétés à régler et les principales méthodes que nous pouvons leur appliquer.

principales classes de contrôles	principales propriétés communes	principales méthodes communes
`Label` (étiquette)	`Value` (valeur)	`SizeToFit` (calculer la bonne taille)
`TextBox` (zone de texte)	`BackColor` (couleur de fond)	`Move(X, Y, L, H)` (déplacer)
`CheckBox` (case à cocher)	`ForeColor` (couleur de texte)	`SetFocus` (y positionner le curseur)
`ComboBox` (liste déroulante)	`Font` (police de caractère)	`Requery` (mettre à jour le contenu)
`ListBox` (zone de liste)	`Locked` (verrouillé)	
`CommandButton` (bouton)	`Enabled` (actif)	
`SubForm` (sous-formulaire)	`Visible`	

TAB. 56 – *Objets importants en VBA pour Access*

Pour obtenir une liste exhaustive des objets disponibles avec VBA pour Access, le lecteur est invité à utiliser l'explorateur d'objet (figure 106) pour accéder à l'aide Visual Basic for Applications qui donne la liste des membres spécifiques à chaque classe.

Par ailleurs, les principaux événements communs à tous ces objets sont donnés dans le tableau 57 (d'autres événements sont visibles sur la figure 108).

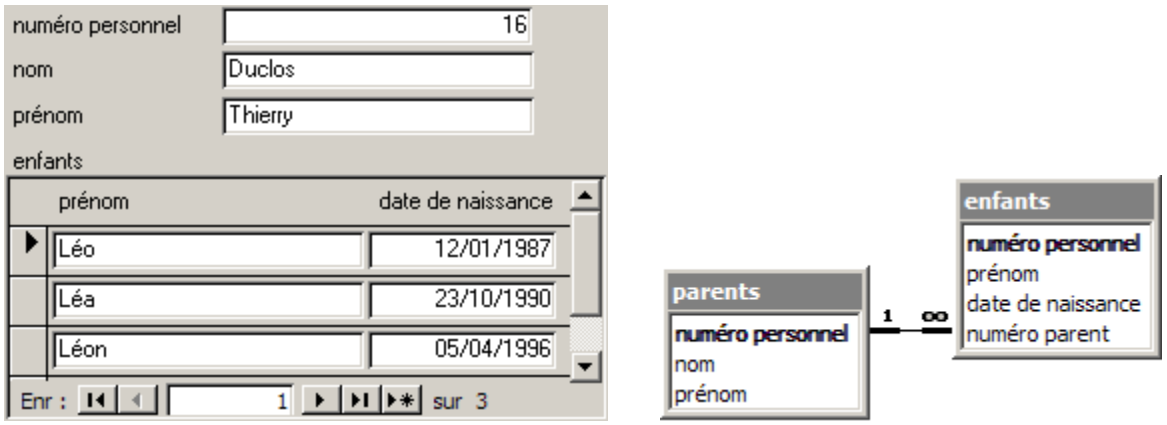
principaux événements communs	
`Click` (clic sur le contrôle par la souris)	`DblClick` (double clic)
`Enter` (entrée dans le contrôle)	`Exit` (sortie)
`BeforeUpdate` (avant saisie d'une valeur)	`AfterUpdate` (après saisie)

TAB. 57 – *Événements VBA importants pour Access*

À chaque événement sur un contrôle dans un formulaire (ou un état), nous pouvons associer une procédure VBA privée[44], programmée dans le module du formulaire en question. Le nom de cette procédure comporte le nom du contrôle et le nom de l'événement :

```
1  Private Sub enfants_Sous_formulaire_Enter()
2  ' après une tentative d'entrée dans le sous-formulaire des enfants
3
4    If IsNull(numéro_personnel) Then         ' si le parent n'est pas encore saisi
5      nom.SetFocus                           ' alors le curseur est re-positionné
6      enfants_Sous_formulaire.Locked = True  ' et le sous-formulaire est verrouillé
7    Else                                     ' sinon, le parent possède un numéro
8      enfants_Sous_formulaire.Locked = False ' le sous-formulaire est déverrouillé
9    End If
10
11 End Sub
```

Grâce à la procédure ci-dessus, l'utilisateur du formulaire de la figure 107(a) ne peut pas se tromper en saisissant d'abord les enfants sans avoir saisi le parent au préalable, ce qui n'est pas permis par le schéma relationnel de la figure 107(b), car la colonne `numéro parent` est déclarée `Null interdit`.



(a) sous-formulaire des enfants bloqué tant que le parent n'est pas saisi

(b) schéma relationnel

FIG. 107 – *Formulaire et sous-formulaire dont la clé étrangère doit être non vide*

Remarquons que lorsque le sous-formulaire est verrouillé par la procédure VBA, il n'est pas vidé, car si l'utilisateur s'est trompé, alors il peut retrouver dans le sous-formulaire déverrouillé, les enfants déjà

44. heureusement que ces procédures sont privées, car sinon, deux formulaires qui comporteraient deux contrôles homonymes entreraient en conflit

saisis (il faut toujours penser qu'un utilisateur peut revenir en arrière).

Notons enfin que le schéma de la figure 107(b) convient pour cet exemple, mais pas dans l'absolu, car entre les tables `enfants` et `parents` il devrait y avoir une liaison plusieurs-à-plusieurs. Mais avec une liaison plusieurs-à-plusieurs, il ne serait pas possible de saisir les enfants dans un sous-formulaire du formulaire des parents. Pour s'en persuader, le lecteur est invité à consulter de nouveau la section 5.4 page 75, dédiée aux sous-formulaires.

7.2.2 Formulaires et états

Un formulaire ou un état est lui-même un objet qui possède notamment les événements `Open` (ouverture) et `Close` (fermeture). À nouveau, si nous le désirons, nous pouvons associer des actions spécifiques au déclenchement de ces événements :

```
1 Private Sub Form_Open(Cancel As Integer)
2   MsgBox("Bienvenue dans ce formulaire")
3 End Sub
```

```
1 Private Sub Report_Close()
2   MsgBox("À bientôt dans cet état")
3 End Sub
```

Pour ne pas se tromper sur le nom de ces procédures et de leurs arguments éventuels, il est conseillé de laisser Access créer lui-même l'en-tête en passant par l'onglet `Événement` de la fenêtre des propriétés du formulaire (figure 108) puis en cliquant sur le petit bouton ... correspondant au bon événement.

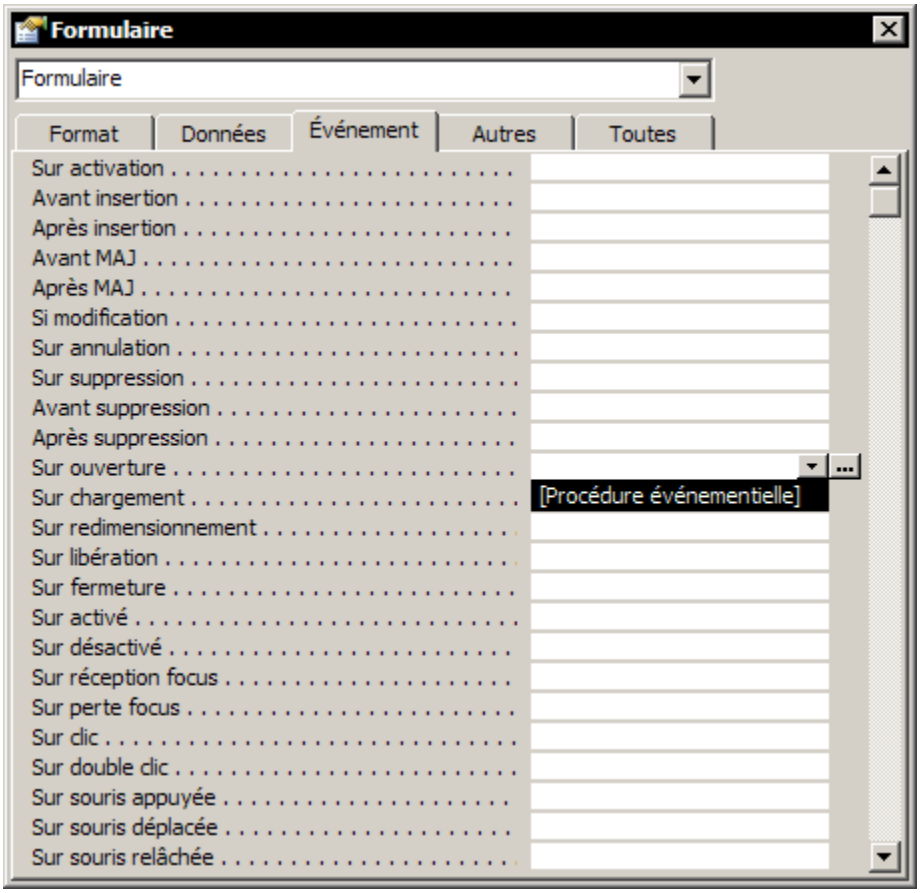


FIG. 108 – *Onglet `Événement` de la fenêtre des propriétés du formulaire*

La zone détail est également un objet qui possède, lui, l'événement `Print` dont nous pouvons nous servir pour griser une ligne sur deux :

```
Private Sub Détail_Print(Cancel As Integer, PrintCount As Integer)

  Const gris = 13816530, blanc = 16777215

  Détail.BackColor = IIf(Détail.BackColor = blanc, gris, blanc)

End Sub
```

Cette procédure est parfois utile pour un état ou un sous-formulaire comportant beaucoup de lignes, car elle en facilite la lecture. Nous pouvons donc l'utiliser pour l'annuaire de la section 6 (figure 109).

Annuaire des employés

identité	*téléphone*	*domicile*
A		
Paul Acortia	04 93 ...	18 rue ...
Marie Andrie	03 27 ...	7 boulevard ...
B		
Arthur Boulet	02 41 ...	64 avenue ...

FIG. 109 – *Annuaire comportant une ligne grisée sur deux*

Il ne s'agit pas seulement d'une fonctionnalité esthétique, mais véritablement d'un confort de lecture indispensable, surtout lorsque l'état comporte des lignes successives très rapprochées.

7.2.3 Collections

Un formulaire Access est un objet de la classe `Form`. À ne pas confondre avec la collection `Forms` qui contient tous les formulaires ouverts de la base de données.

Dans le même esprit, un formulaire est une collection de contrôles. Le nom complet du contrôle `Titre` appartenant au formulaire `FormulaireLivres` est `Forms!FormulaireLivres!Titre`, car l'opérateur `!` permet d'accéder à un élément particulier dans une collection.

Le nom complet de la valeur de ce contrôle est donc `Forms!FormulaireLivres!Titre.Value` (ce qui nous sera utile dans la section 8.1 dédiée aux requêtes paramétrées).

Par ailleurs, un contrôle de type zone de liste (`ListBox`) à choix multiple (figure 75 page 74), ne possède pas une valeur unique (`Value`), mais une collection de valeurs sélectionnées. Il s'agit de la collection `ItemsSelected`.

Pour traiter les valeurs sélectionnées dans une liste de sélection multiple, il est nécessaire de faire une boucle sur les éléments de sa collection `ItemsSelected`, mais il s'agit d'une boucle `For` spéciale :

```
1 Dim item As Variant  ' pour une fois, l'indice de boucle est de type Variant
2
3 For Each item In ListeSelectionMultiple.ItemsSelected
4
5   MsgBox(ListeSelectionMultiple.ItemData(item))
6
7 Next item
```

Le tableau `ItemData` contient les données de la colonne liée de la liste `ListeSelectionMultiple`. La boucle ci-dessus affiche donc une série de `MsgBox`, chacune contenant une donnée sélectionnée dans la liste.

La syntaxe de cette nouvelle boucle `For` est donnée dans le tableau 58.

parcours d'une collection
`For Each` variant `In` collection instructions `Next` variant

TAB. 58 – *Boucle sur les éléments d'une collection*

`Variant` est un type de données joker, qui remplace n'importe quel type de données (ici, il remplace le type `Integer`, c'est la collection qui veut ça).

Ce genre de boucle nous sera indispensable dans le traitement SQL des zones de listes à choix multiples (section 8.4).

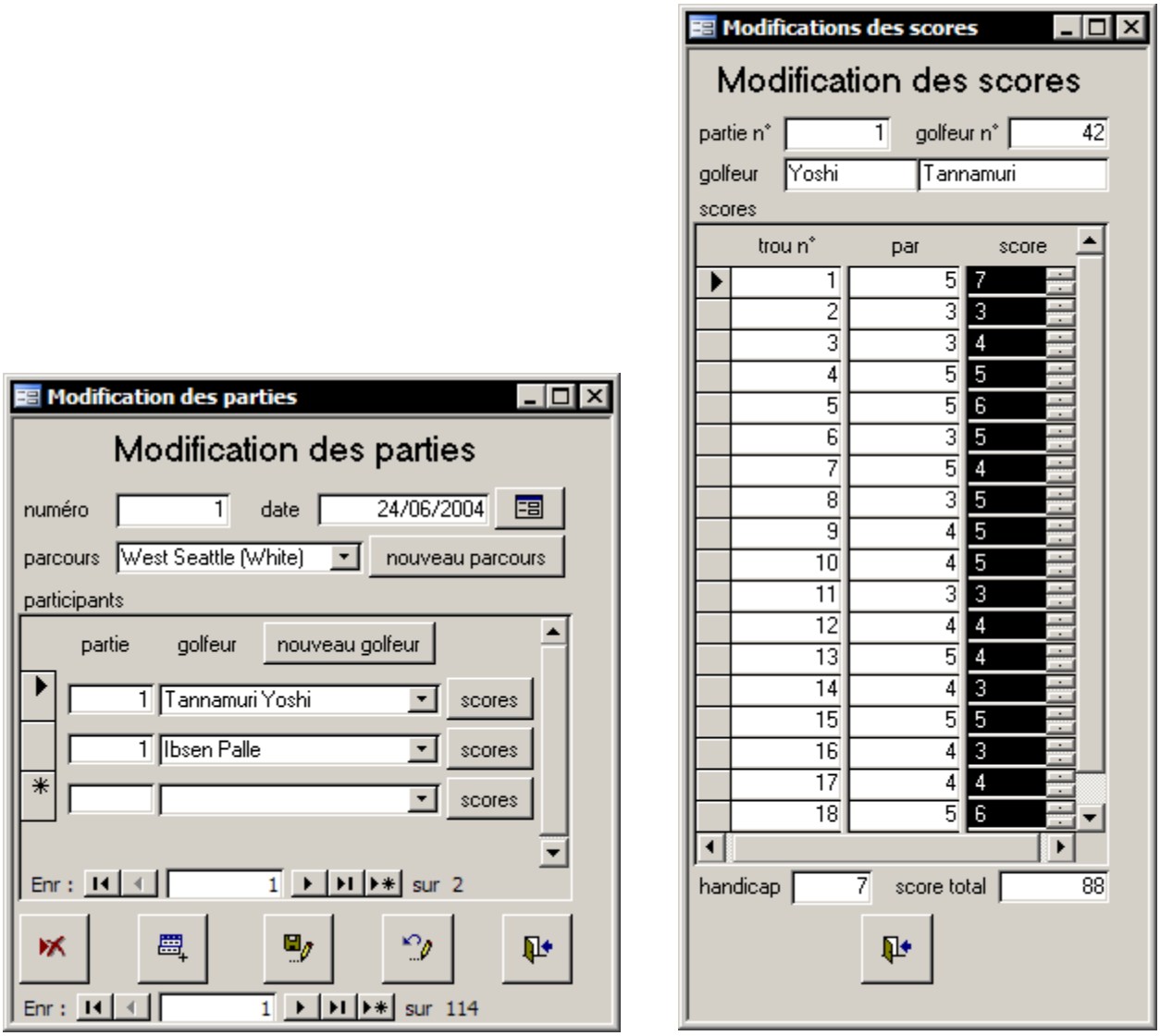
7.2.4 L'objet `DoCmd`

`DoCmd` est un objet VBA qui permet d'exécuter une commande, comme par exemple fermer un formulaire. Le code suivant est celui des deux boutons quitter des deux formulaires de la figure 110 :

```
Private Sub BoutonQuitter_Click()

  DoCmd.Close  ' méthode sans argument

End Sub
```



(a) formulaire appelant

(b) formulaire appelé

FIG. 110 – *Formulaire muni de boutons programmés en VBA et formulaire appelé*

`DoCmd` sert aussi à la programmation de tous les boutons (`supprimer`, `dupliquer`, `sauvegarder`, `annuler`) en bas de la figure 110(a) et le lecteur est invité à consulter cette programmation générée automatiquement par l'assistant `Bouton de commande` (figure 111).

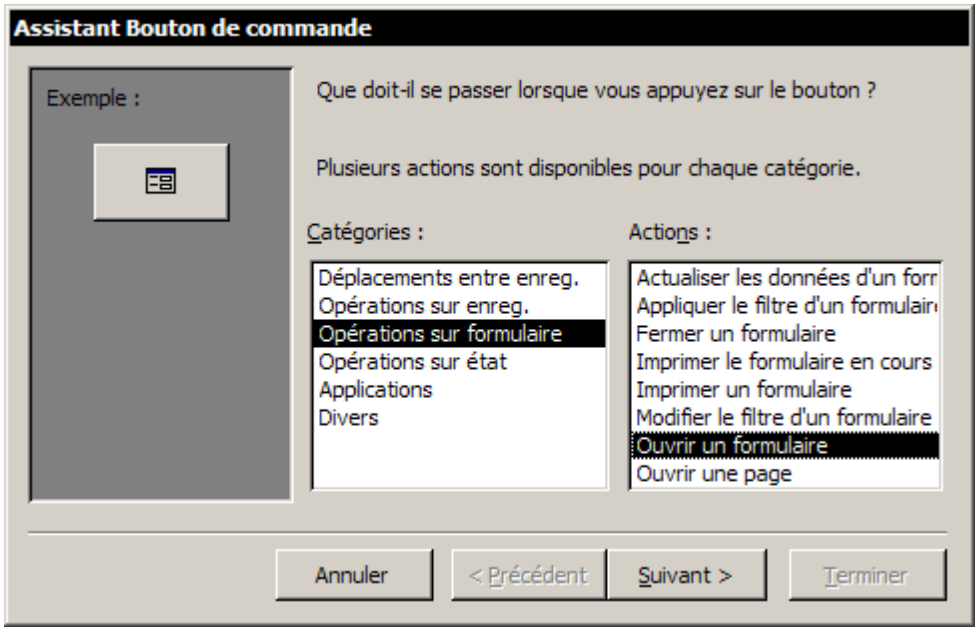


FIG. 111 – *Assistant de création d'un bouton de commande*

`DoCmd` permet aussi d'exécuter une requête d'action (section 4 page 51). Le code suivant exécute la requête de (re-)création de table `nombre livres par auteur` (que nous avions programmée dans le tableau 43 page 59).

```
Private Sub BoutonExecuterRequete_Click()

  DoCmd.OpenQuery "nombre livres par auteur"
  ' l'argument est le nom de la requête, entre guillemets

End Sub
```

Plus fréquemment, `DoCmd` est utilisé pour ouvrir un autre formulaire. C'est le cas du bouton `nouveau parcours` de la figure 110(a) qui ouvre le formulaire de la figure 120 page 126 et dont la programmation de l'événement `clic` est la suivante :

```
Private Sub BoutonNouveauParcours_Click()

  Dim CritereLien As String

  CritereLien = ""
  ' pour ce bouton, il n'y pas de lien à établir
  ' entre le formulaire appelant et le formulaire appelé

  DoCmd.OpenForm "SaisieNouveauParcours", , , CritereLien
  ' l'argument 1 est le nom du formulaire à ouvrir
  ' les arguments 2 et 3 sont inutiles
  ' l'argument 4 est vide, mais il faut le préciser

End Sub
```

Le bouton `nouveau golfeur` de la figure 110(a) fonctionne selon le même principe. Nous l'avons placé dans l'en-tête du sous-formulaire, car il n'est pas nécessaire de le répéter autant de fois que la zone détail.

Par contre, le bouton `score` qui ouvre le formulaire de la figure 110(a) doit être placé dans la zone détail, afin de caler le formulaire appelé sur le bon golfeur. La programmation de ce bouton fait d'ailleurs apparaître un critère de lien entre les formulaires appelant et appelé. Ce critère s'assure que le numéro de partie et le numéro de golfeur sont les bons (le schéma relationnel est celui de la figure 31 page 28) :

```
Private Sub BoutonScore_Click()

  Dim CritereLien As String

  CritereLien = "[numéro partie] =" & NumeroPartie.Value & _
    " AND [numéro golfeur] = " & NumeroGolfeur.Value

  DoCmd.OpenForm "ScoreModification", , , CritereLien

End Sub
```

Dans cet exemple, les contrôles `NumeroPartie` et `NumeroGolfeur` sont les contrôles du détail du sous-formulaire de la figure 110(a). La zone de texte `NumeroPartie` ne peut donc pas être supprimée, même si elle ne sert à rien d'autre. La liste déroulante `NumeroGolfeur` affiche l'identité du golfeur, mais sa valeur VBA est bien le numéro du golfeur (pour s'en persuader, consulter la section 5.3.3 page 70, dédiée aux listes déroulantes).

Dernier exemple, le bouton à côté de la date de partie sur la figure 110(a) ouvre le formulaire de la figure 112 qui est réduit à un contrôle ActiveX calendrier (que nous avions déjà présenté dans la section 5.3.1 page 66) permettant de saisie la date de manière plus ergonomique.

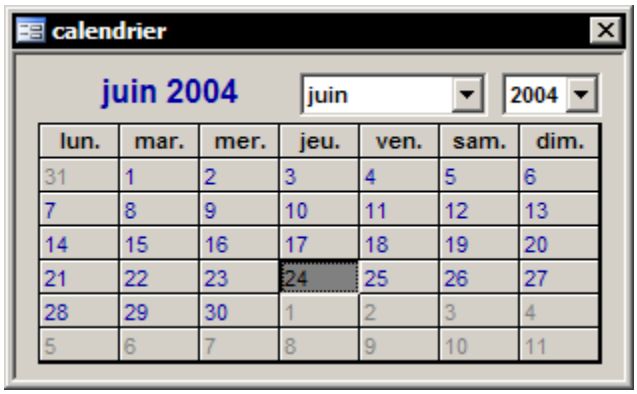


FIG. 112 – *Formulaire calendrier généraliste*

Ce formulaire `calendrier` est généraliste, au sens où il est censé fonctionner pour tous les autres formulaires dans lesquels il est nécessaire de saisir une date. Nous ne savons donc pas à l'avance à quelle colonne la valeur saisie dans le calendrier doit être attribuée. C'est la programmation du bouton qui l'appelle qui décide de cette colonne :

```
Private Sub BoutonDatePartie_Click()
  ' cette procédure (très simple) figure dans le module du formulaire appelant

  Dim CritereLien As String
  CritereLien = ""

  DoCmd.OpenForm "calendrier", , , CritereLien

  Forms!calendrier.RecordSource = Me.RecordSource
  ' le formulaire calendrier doit avoir la même requête source
  ' que le formulaire qui l'appelle (pour la source contrôle qui suit)
  ' Me (moi, en anglais) est le formulaire appelant

  Forms!calendrier!Calendar1.ControlSource = DatePartie.ControlSource
  ' le contrôle Calendar1 doit posséder la même source contrôle
  ' que le contrôle dans laquelle la date doit être saisie

End Sub
```

Il s'agit-là de la première fois que nous changeons la propriété `RecordSource` d'un formulaire et la propriété `ControlSource` d'un contrôle, depuis le code VBA d'un autre formulaire. C'est d'ailleurs pour cette raison que nous devons préciser le nom complet de ces objets.

Cette solution repose tout de même sur l'hypothèse qu'à un instant donné, un seul utilisateur n'a besoin du formulaire `calendrier`. Cette hypothèse est réaliste dans une base de données Access.

Notons enfin que, dans la section 8.4 page 129, nous utilisons beaucoup l'objet `DoCmd` pour exécuter du code SQL.

7.3 Erreurs et débogage

Lorsque nous programmons en Visual Basic for Applications, nous pouvons commettre trois types d'erreurs :

- des erreurs de syntaxe, pour lesquelles :
  - l'interpréteur VBA nous prévient quand nous passons à la ligne, en ce qui concerne les erreurs sur les mots-clés et la ponctuation (ce qui permet de les corriger à la volée) ;
  - Access nous le signale à l'exécution, en ce qui concerne les erreurs sur les noms de constantes, de variables, de procédures et de fonctions (noms qui sont faciles à corriger grâce à la complétion automatique : `CTRL + espace`, rappelons-le) ;
- des erreurs de logique (une routine ne fait pas ce que nous croyons lui avoir demandé), auquel cas il faut déboguer ;
- des erreurs soulevées par un objet Access qui n'est pas dans l'état attendu (un formulaire non ouvert, par exemple), auquel cas il faut piéger ces erreurs.

7.3.1 Déboguer

Pour comprendre pourquoi une procédure ou une fonction ne fait pas ce que nous en attendons, nous pouvons :

- exécuter le code pas à pas (c'est-à-dire instruction par instruction) avec le bouton ;
- poser des points d'arrêt dans le code aux endroits que nous soupçonnons de générer des erreurs, en cliquant dans la marge afin de faire apparaître un point rouge[45] tel que sur la figure 113.

```
Private Sub BoutonDatePartie_Click()
  ' cette procédure (très simple) figure dans le module du formulaire appelant

  Dim CritereLien As String
  CritereLien = ""

  DoCmd.OpenForm "calendrier", , , CritereLien

  Forms!calendrier.RecordSource = Mee.RecordSource
  ' le formulaire calendrier doit avoir la même requête-source
  ' que le formulaire qui l'appelle (pour la source contrôle qui suit)

  Forms!calendrier!Calendar1.ControlSource = DatePartie.ControlSource
  ' le contrôle Calendar1 doit posséder la même source contrôle
  ' que le contrôle dans laquelle la date doit être saisie

End Sub
```

FIG. 113 – *Position d'un point d'arrêt dans le code VBA*

45. point rouge qu'il suffit de cliquer à nouveau pour le faire disparaître

Dans les deux cas, l'exécution est suspendue et nous pouvons regarder le contenu des variables et des propriétés des objets, afin de trouver l'erreur à l'aide des techniques suivantes :

- la variable ou la propriété sous le pointeur de la souris affiche sa valeur en info-bulle (au bout de quelques secondes) ;
- la fenêtre des variables locales (figure 114) donne la liste des variables et des objets (avec leurs propriétés) utilisés au moment de l'interruption du code (y compris l'objet `Me`) ;

```
Private Sub BoutonDatePartie_Click()
  ' cette procédure (très simple) figure dans le module du formulaire appelant

  Dim CritereLien As String
  CritereLien = ""

  DoCmd.OpenForm "calendrier", , , CritereLien

  Forms!calendrier.RecordSource = Mee.RecordSource
  ' le formulaire calendrier doit avoir la même requête-source
  ' que le formulaire qui l'appelle (pour la source contrôle qui suit)

  Forms!calendrier!Calendar1.ControlSource = DatePartie.ControlSource
  ' le contrôle Calendar1 doit posséder la même source contrôle
  ' que le contrôle dans laquelle la date doit être saisie

End Sub
```

Variables locales

golf.Form_parties_modification.BoutonDatePartie_Click

Expression	Valeur	Type
Me		Form_parties_modification/Form_parties_modification
CritereLien	""	String
Mee	Vide	Variant/Empty
DatePartie	Vide	Variant/Empty

FIG. 114 – *Fenêtre des variables locales*

La ligne surlignée en jaune est symptomatique du fait que VBA est en mode débogage. Pour quitter le mode débogage, il suffit de cliquer sur le bouton réinitialiser .

– en cliquant droit sur la fenêtre des espions (figure 115), nous pouvons ajouter un espion sur une variable ou un propriété pour en suivre le contenu au fil des interruptions de code ;

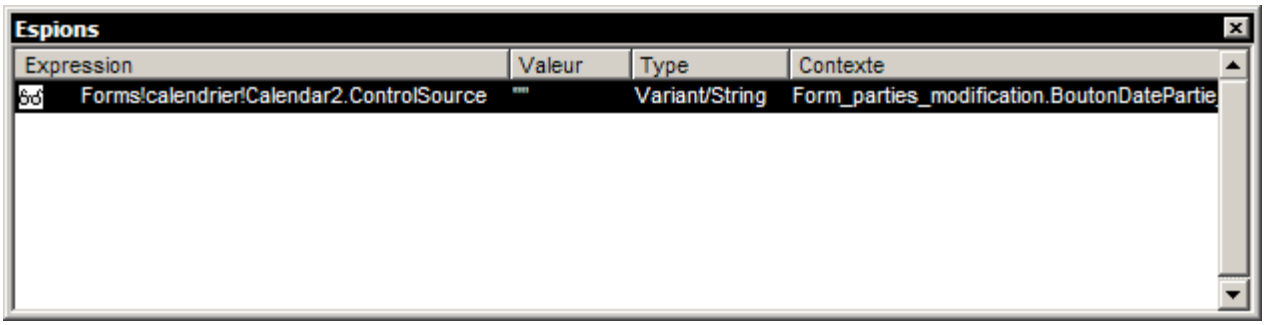


FIG. 115 – *Fenêtre des espions*

– la fenêtre d'exécution (figure 116) permet de demander le contenu d'une variable ou d'une propriété, même extérieure.

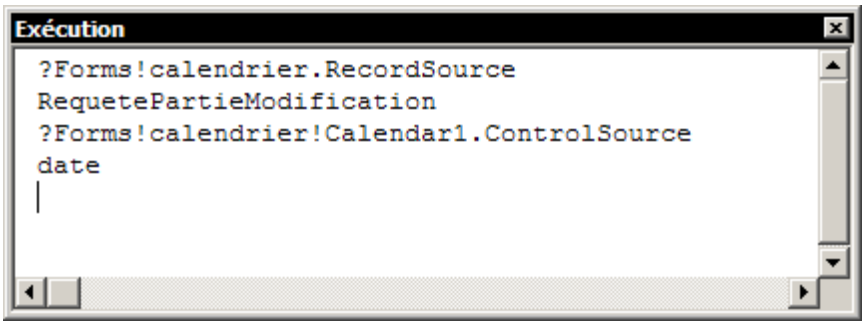


FIG. 116 – *Fenêtre d'exécution*

Par exemple, en tapant la question suivante dans la fenêtre d'exécution :

```
1 ?Forms!calendrier!Calendar1.ControlSource
```

nous obtenons en guise de réponse (figure 116), le contenu de la propriété dont le nom complet a été énoncé derrière le curseur (qui est, pour cette fenêtre, en forme de point d'interrogation).

Signalons enfin qu'une autre technique générique de débogage, consiste à ne pas utiliser les outils du débogueur VBA, mais à afficher le contenu d'une variable ou d'une propriété à l'aide de l'instruction `MsgBox` ou de l'instruction `Debug.Print`, qui fonctionne comme une `MsgBox` sans option, mais qui a l'avantage d'imprimer tous les messages d'erreur dans la fenêtre d'exécution. Cette technique, quoique simpliste, est tout à fait valable pour corriger les erreurs de logique dans un code.

7.3.2 Piéger les erreurs

Dans un code VBA exempt d'erreur de syntaxe et d'erreur de logique, il se peut encore qu'un objet Access ne se trouve pas dans l'état attendu. Il peut s'agir d'une division par zéro, d'un fichier impossible à ouvrir ou d'un formulaire inaccessible.

Dans ce cas l'événement `Error` est soulevé. Pour un tel événement, nous disposons de l'objet `Err` qui contient alors des informations sur l'erreur la plus récente, notamment, son numéro et sa description.

Dans chaque nouvelle routine programmée, il est conseillé de piéger l'événement `Error`, ne serait-ce que pour en afficher la description. Access le fait automatiquement lorsque l'un de ses assistants créé une nouvelle routine (pour un bouton par exemple). La structure de la routine est alors de cette forme :

```
1  Private Sub BoutonDatePartie_Click()
2  On Error GoTo Err_BoutonDatePartie_Click
3
4    ...          ' instructions pour l'exécution normale
5
6  Exit_BoutonDatePartie_Click:
7
8    Exit Sub
9    ' fin de la procédure, pour éviter que la partie suivante,
10   ' qui gère les erreurs, ne s'exécute en l'absence d'erreur
11
12 Err_BoutonDatePartie_Click:
13 ' cette partie doit être placée derrière le Exit
14
15   Debug.Print "erreur n° " & Err.Number & _
16     " dans MonBouton_Click : " & Err.Description
17   ' affichage de l'erreur dans le débogueur
18
19   Resume Exit_BoutonDatePartie_Click
20   ' pour sortir de la procédure après l'affichage
21
22 End Sub
```

L'utilisation de `Debug.Print` permet d'afficher le message d'erreur (destiné au développeur) dans la fenêtre d'exécution (figure 116) lors du débogage, ce qui, contrairement à l'instruction `MsgBox`, a l'avantage de ne pas perturber l'utilisateur.

L'instruction `Resume` est équivalente à l'instruction `GoTo` (que nous avions déjà rencontrée dans la section 7.1.6 page 108). Par contre, l'utilisation de `Resume Exit_MonBouton_Click` est une solution pessimiste pour piéger les erreurs, car à la moindre erreur, l'exécution du code restant de la procédure est annulée. Nous pouvons donc remplacer cette ligne par une autre ligne plus optimiste, comme :

- `Resume Next` qui passe à l'instruction suivante de celle qui a soulevé l'erreur (ce qui peut être assez hasardeux) ;
- `Resume 0` qui ré-essaye indéfiniment l'instruction fautive jusqu'à ce qu'elle fonctionne (ce qui peut provoquer une boucle infinie).

Notons que nous pouvons également utiliser les instructions `On Error GoTo Next` et `On Error GoTo 0`, n'importe où dans le programme afin de gérer différemment les erreurs de certains blocs de lignes. Nous invitons le lecteur à consulter l'aide d'Access sur ce sujet pour découvrir un exemple très parlant concernant l'ouverture d'un fichier extérieur. C'est également l'occasion de découvrir la possibilité que nous avons de lever nos propres erreurs à l'aide de la méthode `Err.Raise`.

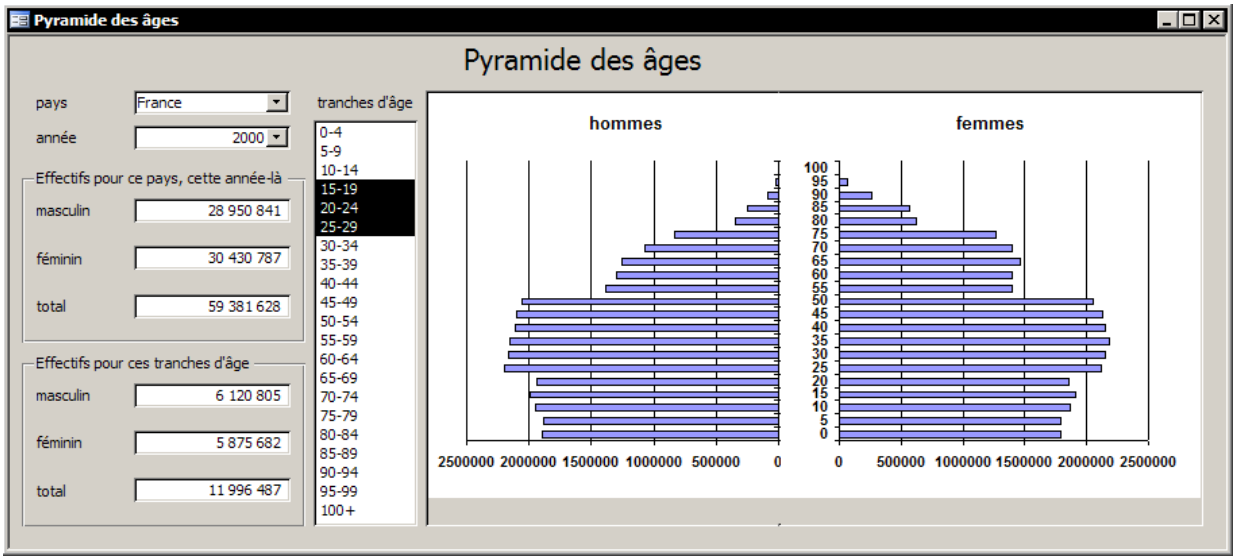
8 VBA et SQL

Le développeur VBA est souvent amené à gérer les données Access *via* du code SQL ou, à l'inverse, une requête SQL peut avoir besoin de la valeur d'un objet VBA.

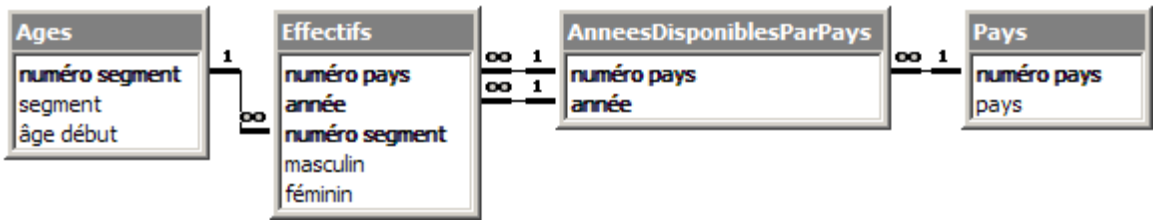
8.1 Requêtes paramétrées

Une requête Access peut faire appel à la valeur d'un contrôle qui se trouve dans un formulaire ouvert. Cette fonctionnalité est très utile pour établir des listes déroulantes dont le contenu dépende de la valeur d'une autre liste saisie précédemment.

Par exemple, sur le formulaire `Pyramide des âges` de la figure 117(a), la liste déroulante des années est calculée en fonction du pays choisi précédemment dans la liste `pays`. En effet, pour la France nous disposons des années 1990, 1995, 2000 et 2005, tandis que pour la Suisse nous ne disposons de données que pour 2000 et 2005. Nous stockons donc les années disponibles dans la table `AnneesDisponiblesParPays` (figure 117(b)).



(a) formulaire de consultation



(b) schéma relationnel des tables du formulaire

FIG. 117 – *Consultation de la pyramide des âges d'un pays au cours d'une année*

La requête qui permet de sélectionner les années disponibles par le pays choisi dans le formulaire est celle du tableau 59.

les tables et leurs liaisons	`AnneesDisponiblesParPays`
les colonnes et leur affichage	`année`, `pays` (non affiché)
les critères sur les lignes	`pays = Forms!consultation!Pays.Value`
les tris et leur sens	année (croissant)

TAB. 59 – *Requête paramétrée*

Cette requête est paramétrée, son paramètre étant `Forms!consultation!Pays.Value` (le nom complet des valeurs d'un formulaire est donné dans la section 7.2.3 page 113).

Si ce paramètre est correctement orthographié et si le formulaire est bien ouvert, la requête s'exécute normalement. Sinon, Access demande à l'utilisateur la valeur qu'il désire pour ce paramètre qu'il ne comprend pas (figure 118).

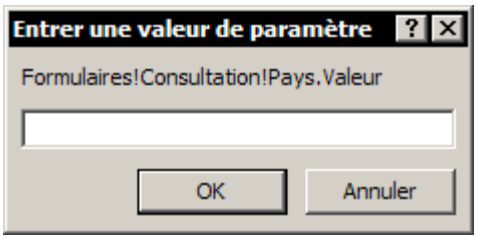


FIG. 118 – *Boîte de dialogue demandant le paramètre de la requête quand le formulaire n'est pas ouvert*

Cette requête peut ensuite servir de contenu au contrôle `Annee` dans le formulaire (la propriété `Contenu` est une notion abordée dans la section 5.3.3 page 70). Mais il faut que ce contenu soit recalculé à chaque fois que la valeur de la liste `Pays` change. Pour cela, nous réagissons à l'événement `AfterUpdate` de ce contrôle *via* la procédure VBA suivante :

```
1  Private Sub Pays_AfterUpdate()
2
3    Annee.Requery
4    Ages.Requery
5
6    EffectifPartielMasculin.Requery
7    EffectifPartielFeminin.Requery
8    EffectifPartiel.Requery
9
10   PyramideHommes.Requery
11   PyramideFemmes.Requery
12
13 End Sub
```

Dans cette procédure, nous voyons également toutes les opérations de mise à jour (méthode `Requery`) des contrôles du formulaire (les deux autres listes, les trois zones de texte calculées et les deux sous-formulaires graphiques), lorsque le pays choisi change. Comme nous pouvons le voir sur cet exemple simple, VBA et SQL peuvent être intimement mêlés pour obtenir un formulaire qui réagit dynamiquement.

8.2 Domaines

Un domaine est un ensemble de lignes issues d'une requête ou d'une table et vérifiant un critère écrit en SQL. En choisissant une colonne particulière d'un domaine, il est possible d'y appliquer une fonction d'agrégation VBA, parmi : `DSum` (`SomDom`), `DCount` (`CpteDom`), `DMin` (`MinDom`), `DMax` (`MaxDom`), `DAvg` (`MoyDom`) ou `DVar` (`VarDom`). La syntaxe commune à toutes ces fonctions est la suivante :

```
1 DCount("nom de la colonne"; "nom de la table ou de la requête"; "critère en SQL")
```

8.2.1 Zone de texte calculée

Par exemple, pour calculer la population féminine totale dans le formulaire `Pyramide des âges` (figure 117, toujours), nous effectuons une somme sur la colonne `féminin` de la table `effectifs`, en prenant garde de sélectionner le bon pays et la bonne année :

```
1 DSum("féminin"; "effectifs"; "[numéro pays] = Forms!consultation!Pays.Value
2                               AND [année] = Forms!consultation!Annee.Value")
```

Cette formule est saisie[46] dans la propriété `Source contrôle` du contrôle `EffectifPartielFeminin` sans oublier de la précéder d'un signe égal `=` (figure 119).

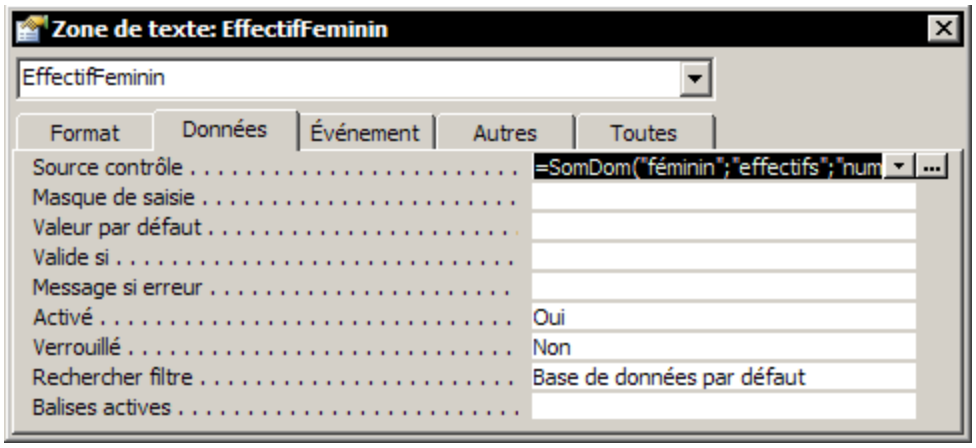


FIG. 119 – *Source contrôle d'un contrôle calculé basé sur une agrégation de domaine*

Nous faisons de même pour l'effectif masculin, puis la formule qui définit la source contrôle de la population totale est tout simplement : `=[EffectifMasculin]+[EffectifFeminin]`. Reste à gérer le recalcul automatique à chaque fois que le pays voulu ou l'année voulue sont modifiés, ce qui est fait à l'aide de la méthode `Requery` (comme précédemment).

46. en utilisant éventuellement le générateur d'expression (dont une capture d'écran est disponible sur la figure 10 page 13)

8.2.2 Critère fondé sur une liste à sélection multiple

Pour les effectifs partiels (toujours sur la figure 117(a)), le domaine est défini par les tranches d'âge choisies par l'utilisateur. Nous devons donc calculer le critère à l'aide d'une boucle `For Each` (que nous avions introduite dans la section 7.2.3 page 113) sur la liste de sélection multiple `Ages` :

```
Dim TranchesSelectionnees As String
' il s'agit d'une variable visible par toutes les routines du module

Private Sub Ages_AfterUpdate()

  Dim NumeroItem As Variant

  TranchesSelectionnees = "0"
  ' il nous faut une valeur initiale sans conséquence pour commencer la chaîne

  For Each NumeroItem In Ages.ItemsSelected

    TranchesSelectionnees = TranchesSelectionnees & ", " & Ages.ItemData(NumeroItem)

  Next NumeroItem

End Sub
```

Cette solution, basée sur une boucle de concaténations, n'est certes pas élégante du point de vue informatique. Mais, cette mise à jour permanente de la chaîne de caractères `TranchesSelectionnees` permet de programmer la fonction suivante, qui sert de source contrôle à la zone de texte `EffectifPartielMasculin` :

```
Public Function PopulationPartielleMasculine() As Long

  If Not IsNull(annee) And Not IsNull(pays) Then

    PopulationPartielleMasculine = _
      DSum("masculin", "effectifs", _
           "[numéro pays] = " & Pays.Value & _
           " AND année = " & Annee.Value & _
           " AND [numéro segment] IN (" & TranchesSelectionnees & ")")

  Else

    PopulationPartielleMasculine = 0

  End If

End Function
```

Nous faisons de même pour l'effectif partiel féminin, puis nous sommons les deux dans l'effectif partiel (tout court). Reste à ajouter les bons `Requery` dans la procédure `Ages_AfterUpdate` pour que ce formulaire réagisse parfaitement.

8.2.3 Sous-numérotation

Les fonctions d'agrégation de domaine peuvent également être servir à gérer la sous-numérotation, qui, rappelons-le, est une opération manuelle (comme nous l'avons vu dans la section 2.10 page 28). Par exemple, dans le sous-formulaire du formulaire `SaisieNouveauParcours` de la figure 120, à chaque nouveau trou, l'utilisateur doit saisir le numéro du trou dans le parcours car il s'agit d'une sous-numérotation (figure 31 page 28).

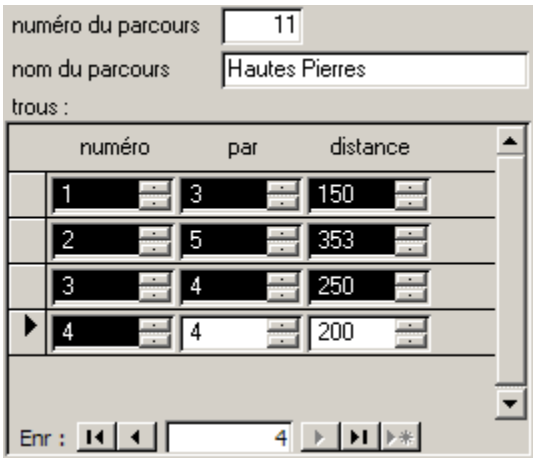


FIG. 120 – *Sous-formulaire muni d'une sous-numérotation*

Outre le fait qu'une sous-numérotation est une valeur numérique dont le contrôle approprié est une toupie (comme dans la section 5.3.4 page 72), il est plus ergonomique de prévoir une valeur par défaut qui est recalculée à la volée grâce à une formule VBA.

Dans notre exemple, pour connaître la valeur suivante du `numéro trou dans parcours`, il suffit d'ajouter 1 à la valeur maximale de cette colonne pour le `numéro parcours` concerné. C'est pourquoi, nous remplissons la propriété `Valeur par défaut` (onglet `Données`) du premier contrôle dans le détail du sous-formulaire avec la formule suivante :

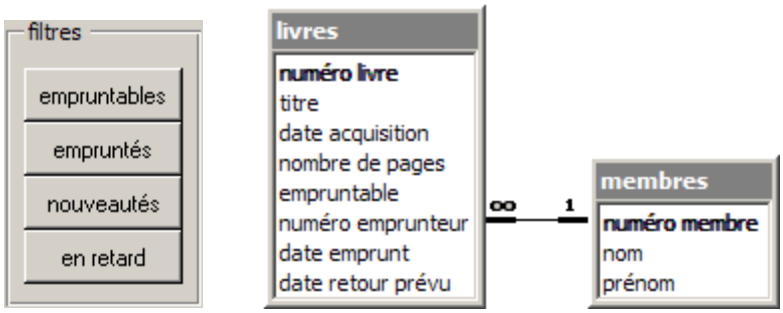
```
1 = 1 + DMax("[numéro trou dans parcours]"; "trous";
2   "[numéro parcours] = Forms!SaisieNouveauParcours!NumeroParcours.Value")
```

Comme il ne s'agit que d'une valeur par défaut et pas d'une valeur définitive (l'utilisateur peut tout à fait décider d'une autre valeur), il n'est pas nécessaire de recourir à une méthode `Requery` pour la mettre à jour. Il se peut donc que cette sous-numérotation laisse des numéros intermédiaires inutilisés, ce que la valeur par défaut proposée ci-dessus ne peut corriger. À charge pour l'utilisateur de combler les vides, s'il le désire.

Cette solution repose tout de même sur l'hypothèse que le sous-formulaire n'est utilisé qu'avec le formulaire principal `SaisieNouveauParcours`. Dans le cas contraire, il suffit de cacher un contrôle `NumeroParcours` dans l'en-tête du sous-formulaire lui-même, afin de retrouver le bon numéro de parcours.

8.3 Filtres

Un filtre permet de sélectionner les enregistrements qui défilent dans un formulaire (notamment les formulaires de consultation et de modification). Nous avons vu qu'il était possible à tout moment à l'utilisateur de définir son propre filtre et de l'appliquer (section 5.7 page 80). Mais, il est par ailleurs possible au développeur d'un formulaire de pré-définir un ou plusieurs filtres pertinents et de les associer à des boutons (figure 121).



(a) boutons à bascule

(b) rappel des colonnes

FIG. 121 – *Filtres prédéfinis en VBA pour la consultation des livres*

Les boutons de la figure 121(a) sont des boutons à bascule (créés *via*), afin qu'ils restent soit enfoncés, soit relâchés. Nous autorisons l'utilisateur à enfoncer plusieurs de ces boutons (qui sont regroupés dans un cadre qui n'a pas signification de groupe d'options, contrairement au cadre de la section 5.3.2 page 68).

Lorsque l'un d'eux est enfoncé, sa valeur vaut -1 (c'est VBA qui veut ça) et son filtre doit être pris en compte. Lorsqu'il est relâché, sa valeur vaut 0 et son filtre doit être retiré. Pour cela, nous concaténons les critères des quatres boutons, qui sont définis dans quatre chaînes de caractères déclarées globalement dans le module :

```
Option Compare Database
Option Explicit

Const Début As String = "True = True"
' il nous faut un critère sans conséquence pour commencer la chaîne

Dim Critère1, Critère2, Critère3, Critère4 As String

Private Sub BasculeEmpruntables_Click()

  If BasculeEmpruntables.Value = -1 Then
      Critère1 = " AND empruntable = True"
  Else
      Critère1 = ""
  End If

  Me.Filter = Début & Critère1 & Critère2 & Critère3 & Critère4
  Me.FilterOn = True

End Sub
```

`Me` (moi, en anglais) est le formulaire auquel le module VBA correspond. Cet objet comporte la chaîne `Filter` dans laquelle nous pouvons écrire le critère SQL qui correspond au filtre. Il reste à mettre en application le filtre en positionnant à vrai[47] le booléen `FilterOn` (qui appartient au même objet `Me`).

Tous les événements clic sur les autres boutons de filtre sont programmés de la même manière :

```
Private Sub BasculeEmpruntés_Click()

  If BasculeEmpruntés.Value = -1 Then
    Critère2 = " AND [numéro emprunteur] IS NOT NULL"
  Else
    Critère2 = ""
  End If

  Me.Filter = Début & Critère1 & Critère2 & Critère3 & Critère4
  Me.FilterOn = True

End Sub

Private Sub BasculeNouveautés_Click()

  If BasculeNouveautés.Value = -1 Then
    Critère3 = " AND [date acquisition] >= DateAdd('d',-30,Date())"
  Else
    Critère3 = ""
  End If

  Me.Filter = Début & Critère1 & Critère2 & Critère3 & Critère4
  Me.FilterOn = True

End Sub

Private Sub BasculeRetard_Click()

  If BasculeRetard.Value = -1 Then
    Critère4 = " AND [date retour prévu] < Date()"
  Else
    Critère4 = ""
  End If

  Me.Filter = Début & Critère1 & Critère2 & Critère3 & Critère4
  Me.FilterOn = True

End Sub
```

Par ailleurs, la syntaxe SQL n'autorise pas la syntaxe française des fonctions qu'offre Access dans le générateur d'expression. Par conséquent, la fonction `AjDate("j";-30;Date())` (`j` pour jour) devient `DateAdd('d',-30,Date())` (`d` pour day) dans la programmation VBA.

47. pour désactiver le filtre du formulaire, il suffit de positionner `FilterOn` à faux

8.4 Gérer une liste à sélection multiple

Rappelons qu'à chaque fois que nous employons une zone de liste avec sélection multiple (comme dans la section 5.3.5 page 73), il est impératif de programmer les boutons du formulaire afin de synchroniser cette liste avec la table de jonction à laquelle elle correspond.

Considérons par exemple le schéma relationnel de la figure 122, conçu pour une application dédiée à la saisie de cartons d'invitation pour un mariage.

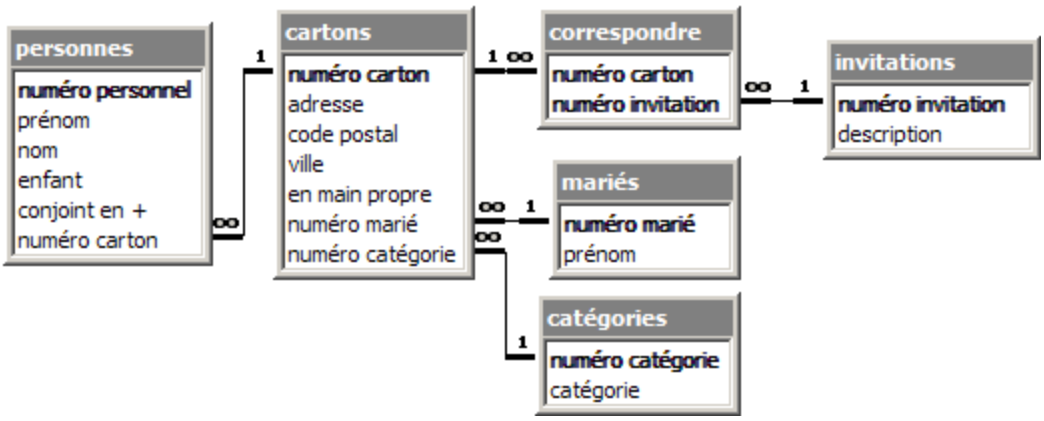


FIG. 122 – *Schéma relationnel pour stocker des cartons d'invitation, avec une liaison plusieurs-à-plusieurs gérée par une liste à sélection multiple*

Sur ce schéma, nous voyons une liaison plusieurs-à-plusieurs entre les tables `cartons` et `invitations` et nous souhaitons que les formulaires de saisie et de modification des cartons puissent remplir également la table de jonction `correspondre`. Pour cela, nous les munissons d'une zone de liste qui contient les colonnes `numéro invitation` et `description` (mais qui n'affiche que la colonne `description`, conformément à nos conseils de la section 5.3.3 page 70).

L'assistant Access qui permet de créer cette zone de liste, n'offre malheureusement pas la possibilité de lier correctement les valeurs multiples choisies dans cette liste à la colonne `numéro invitation` de la table `correspondre`, quand bien même cette colonne fait partie de la requête du formulaire. Il faut donc que nous gérions manuellement cette correspondance.

Pour cela, nous faisons appel à l'objet `DoCmd` (que nous avons introduit dans la section 7.2.4 page 114) et sa méthode `RunSQL` que nous n'avons pas encore utilisée. `RunSQL` permet tout simplement d'exécuter une requête d'action, écrite en SQL et passée, sous la forme d'une chaîne de caractères, en argument de cette méthode.

La syntaxe SQL des requêtes d'action mono-tables est décrite dans la section 4 page 51. Rappelons que nous avons évité d'écrire les requêtes multi-tables en SQL pour ne pas avoir à gérer la syntaxe difficile de la clause `FROM` avec les jointures. Pour l'écriture d'une commande `RunSQL`, nous pouvons également éviter cette écriture SQL multi-table difficile, soit en décomposant le travail table par table, soit en créant au préalable la requête (ou plutôt, la vue, conformément à la section 3.4 page 35) qui effectue le travail de jointure et en n'utilisant qu'elle dans la clause `FROM`.

L'avantage de `RunSQL` par rapport aux requêtes paramétrées, est la possibilité d'adapter la chaîne de caractères qui défini la requête, notamment de concaténer les valeurs VBA pertinentes. En pratique, `RunSQL` est donc la commande la plus utilisée dans VBA.

8.4.1 Dans un formulaire de saisie

Considérons le formulaire de saisie de la figure 123, dont nous devons programmer le bouton de sauvegarde (cela suffit dans un formulaire de saisie, car il ne gère qu'une fiche à la fois, nouvelle qui plus est).

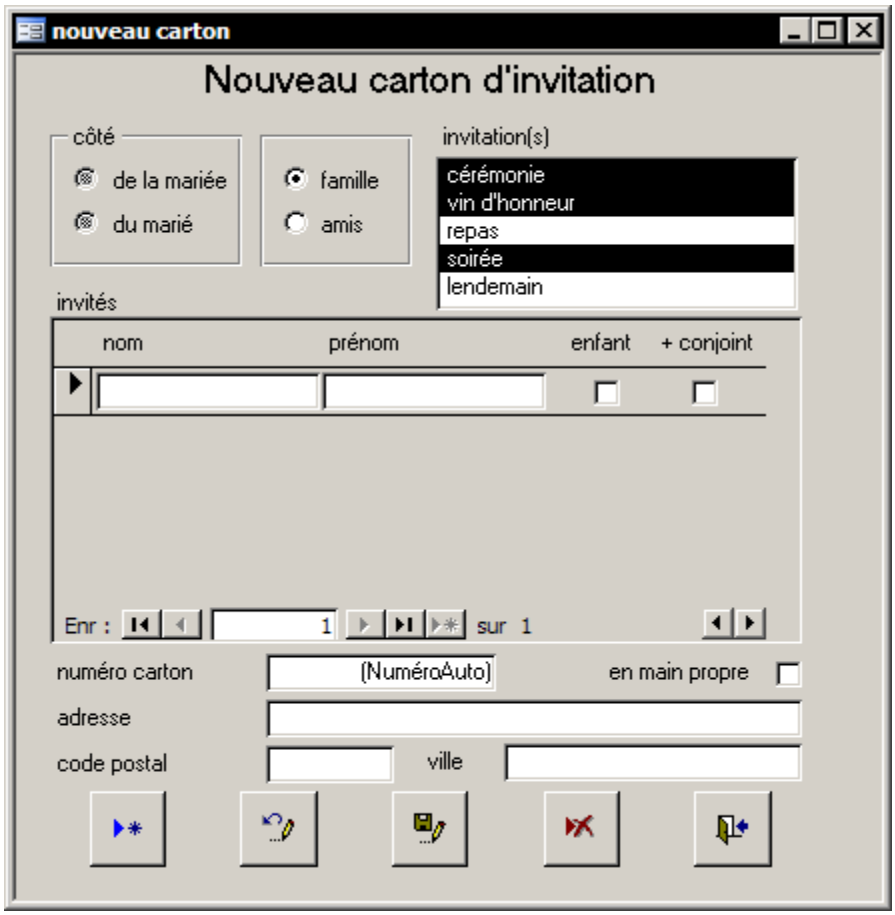


FIG. 123 – *Formulaire de saisie d'un carton de mariage, muni d'une liste à sélection multiple*

Lors du clic sur le bouton de sauvegarde, il faut non seulement enregistrer les renseignements saisis dans la table `cartons`, mais aussi les invitations choisies dans la liste, dans la table `correspondre`. Pour cela, nous parcourons la collection des valeurs sélectionnées par une boucle `For Each` et nous effectuons une insertion SQL avec l'instruction `INSERT INTO` (déjà décrite dans la section 4.1 page 53) que nous pouvons exécuter, dans le code VBA, avec la commande `RunSQL`.

Cette requête `INSERT INTO` est paramétrée, à la fois par le numéro de carton récupéré dans le contrôle `NumeroCarton` et par le numéro numéro d'invitation, récupéré parmi les valeurs sélectionnées dans la liste `ListeInvitations`.

Le code VBA de l'événement clic du bouton de sauvegarde implique donc une série d'insertions élémentaires :

```
Private Sub BoutonSauvegarder_Click()
  ...

  ' sauvegarde prévue initialement dans la table [cartons]
  DoCmd.DoMenuItem acFormBar, acRecordsMenu, acSaveRecord, , acMenuVer70

  ' sauvegarde complémentaire pour la table [correspondre]
  Dim Position As Variant

  For Each Position In ListeInvitations.ItemsSelected

    DoCmd.RunSQL _
      ("INSERT INTO correspondre ([numéro carton], [numéro invitation]) " & _
       "VALUES (" & NumeroCarton.Value & " ," & _
                    ListeInvitations.ItemData(Position) & ")")

    ' rappelons que nous ne pouvons pas passer à la ligne avant d'avoir
    ' clôturé la chaîne de caractères avec une double quote " fermante
    ' il faut donc la couper en autant de morceaux que nécessaire
    ' avec une concaténation ad hoc pour les recoller convenablement

  Next Position

  ' nous ajoutons cette commande pour passer immédiatement à une nouvelle fiche
  DoCmd.GoToRecord , , acNewRec

  ...
End Sub
```

Comme nous insérons une ligne à la fois dans la table correspondre, nous employons l'instruction `INSERT ... VALUES ...` au lieu de l'instruction `INSERT ... SELECT ...`, qui permet d'insérer plusieurs lignes à la fois, mais à partir des données déjà présentes dans la base. La clause `VALUES` est très simple, puisqu'elle introduit, entre parenthèses, les valeurs à insérer dans le même ordre que les colonnes rappelées juste avant.

Le fait de passer immédiatement à un nouveau carton d'invitation, *via* l'instruction `GotoRecord` après l'enregistrement du carton fraîchement saisi, évite au programmeur de devoir gérer un éventuel changement d'avis de l'utilisateur concernant les invitations sélectionnées[48]. Le passage à ce nouveau carton laisse intactes les invitations sélectionnées précédemment dans la liste, cela permet à l'utilisateur d'avoir le moins de modification à apporter si le nouveau carton porte les mêmes invitations que le précédent.

Toutes les listes de sélection multiple peuvent être gérées ainsi. Pour les listes duales, ce n'est pas plus compliqué, mais il faut programmer dans plusieurs procédures (section 8.5). Auparavant, intéressons-nous au travail à effectuer dans un formulaire de modification.

48. c'est un peu arbitraire, il faut bien le reconnaître, mais l'utilisateur a toujours la possibilité de modifier les cartons déjà saisis grâce au formulaire de la section suivante

8.4.2 Dans un formulaire de modification

Dans un formulaire de modification, nous pouvons tricher et remplacer la zone de liste par un sous-formulaire simple, comme sur la figure 124, ce qui a l'avantage de ne nécessiter aucune programmation supplémentaire, pas même pour le bouton sauvegarder.

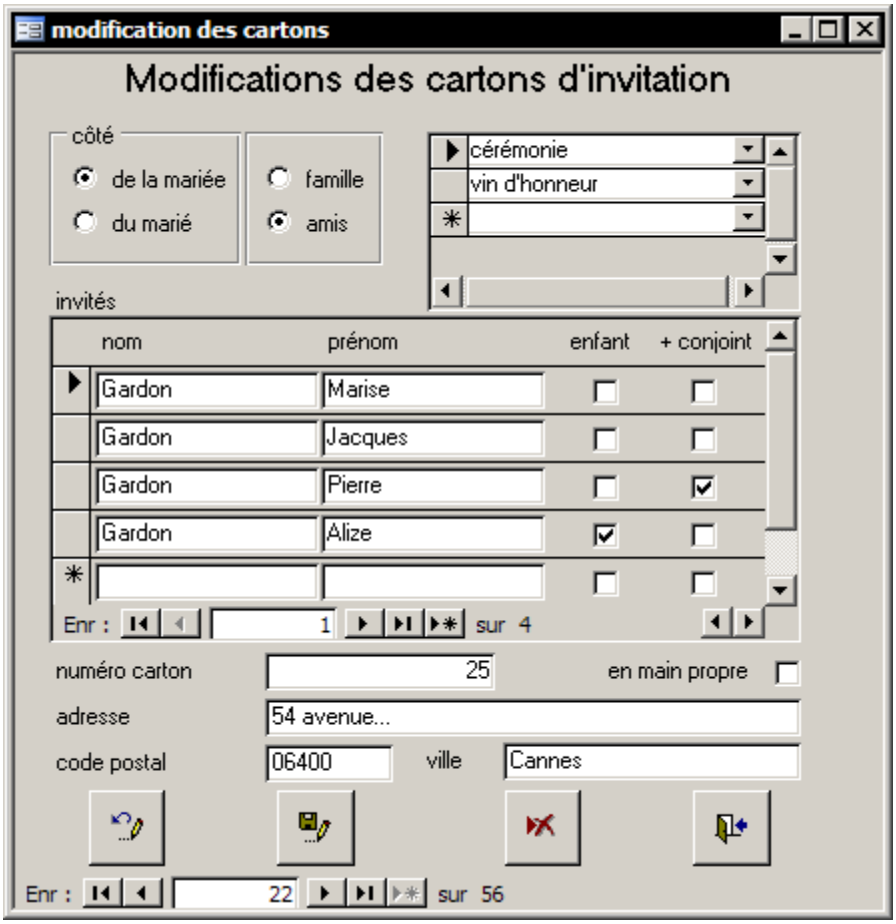


FIG. 124 – *Formulaire de modification dans lequel la zone de liste a été remplacée par un sous-formulaire*

En revanche, le bouton « supprimer » ne fonctionne pas, à cause de ces clés étrangères dont l'intégrité référentielle est assurée. À moins que les liaisons entre les tables `personnes`, `cartons` et `correspondre` aient été toutes les deux déclarées `Effacer en cascade les enregistrements correspondants` (figure 14 page 17).

Mais rappelons que la suppression en cascade n'est pas conseillée, justement parce qu'elle permet de supprimer trop facilement (sur un simple clic) des données dans plusieurs tables. L'absence de suppression en cascade constitue donc toujours un garde-fou précieux. Mais en contre-partie, lorsque l'utilisateur désire vraiment supprimer dans plusieurs tables à la fois, c'est au développeur VBA-SQL de gérer le bon déroulement de cette suppression (comme ci-après).

Dans notre exemple, il nous faut donc programmer la suppression préalable dans la table `correspondre`, ainsi que dans la table `personnes`, avec la procédure suivante :

```
Private Sub BoutonSupprimer_Click()
  ...

  DoCmd.RunSQL("DELETE * FROM correspondre WHERE [numéro carton] = " & _
               NumeroCarton.Value)

  DoCmd.RunSQL("DELETE * FROM personnes WHERE [numéro carton] = " & _
               NumeroCarton.Value)

  ' suppression prévue initialement dans la table 'cartons'
  DoCmd.DoMenuItem acFormBar, acEditMenu, 8, , acMenuVer70
  DoCmd.DoMenuItem acFormBar, acEditMenu, 6, , acMenuVer70

  ...
End Sub
```

Il n'est pas nécessaire, en revanche de programmer les boutons de sauvegarde et d'annulation, car le sous-formulaire des invitations gère correctement les données de la table `correspondre`. Par contre, si nous n'avons pas remplacé la zone de liste de la section précédente par ce sous-formulaire, il faut programmer un certain nombre d'événements supplémentaires :

- le bouton `annuler` doit supprimer les anciennes lignes de la table `correspondre` avec l'instruction `DoCmd.RunSQL("DELETE * FROM correspondre WHERE [numéro carton] = " & NumeroCarton.Value)` de la procédure ci-dessus ;
- le bouton `sauvegarder` doit d'abord supprimer les anciennes lignes de la table `correspondre` (comme le bouton `annuler`), avant d'insérer les nouvelles lignes dans cette même table, comme le bouton `sauvegarder` de la section précédente ;
- le bouton `quitter` peut générer des erreurs, que le développeur à le devoir de gérer afin que l'utilisateur ne soit pas importuné ;
- le plus difficile est de synchroniser la liste à sélection multiple avec les fiches qui défilent dans ce formulaire.

L'emploi du sous-formulaire est donc préférable dans ce type de formulaire de modification. Mais pour le formulaire de saisie de la section précédente, la zone de liste à sélection multiple reste plus ergonomique pour saisir une nouvelle fiche.

8.5 Gérer deux listes duales

Considérons le formulaire de la figure 125, réalisé pour la saisie d'informations relatives à un match de football et concentrons-nous sur la colonne de gauche, dédiée à l'équipe qui joue à domicile.

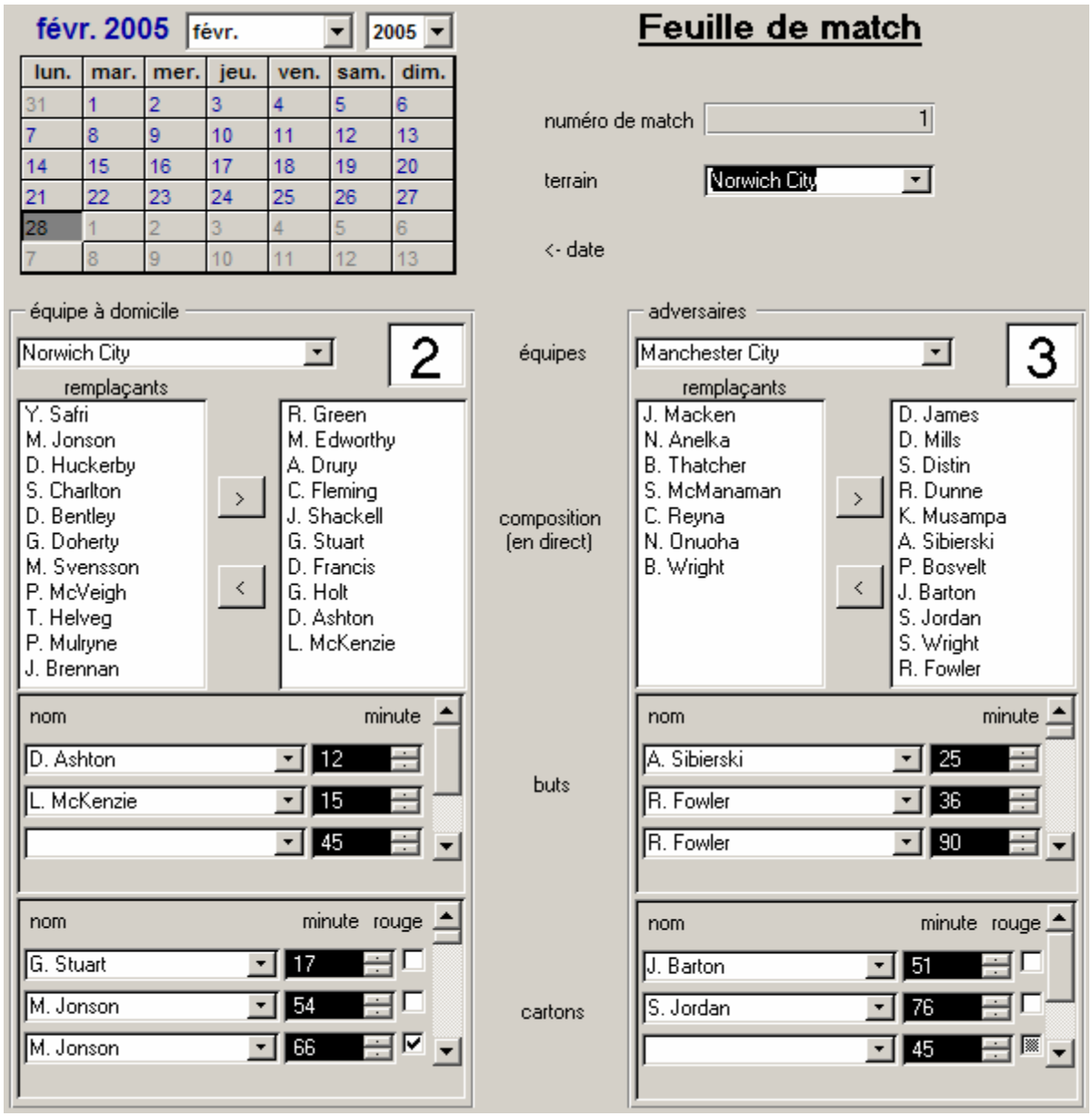


FIG. 125 – *Formulaire de saisie d'une feuille de match*

Pour l'équipe qui joue à domicile, deux listes duales ont été prévues pour gérer, en direct, la composition de cette équipe. Nous devons donc gérer ces quatres contrôles :

- une première zone de liste `RemplacantsDomicile` à sélection multiple, pour les joueurs sur le banc de touche (y compris les joueurs pénalisés par un carton rouge) ;
- une deuxième zone de liste `JoueursDomicile` à sélection multiple, pour les joueurs sur le terrain ;
- un bouton `BoutonVersDroite` qui permet de basculer les remplaçants sélectionnés dans la liste `RemplacantsDomicile` vers la liste `JoueursDomicile` ;
- et un bouton `BoutonVersGauche` pour faire l'inverse, c'est-à-dire basculer les joueurs sélectionnés dans la liste `JoueursDomicile` vers la liste `RemplacantsDomicile`.

La requête source de ce formulaire et les différentes requêtes contenu de ses listes, reposent sur le schéma relationnel de la figure 126.

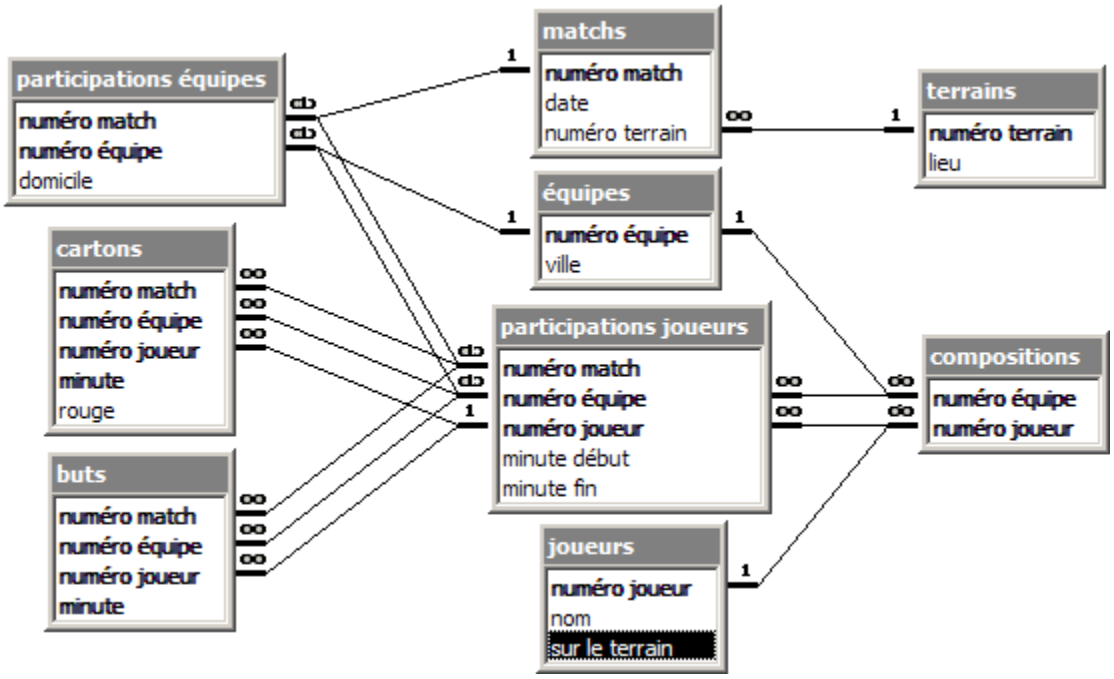


FIG. 126 – *Schéma relationnel du formulaire précédent*

Ce schéma relationnel utilise intensivement la notion de liaison composite, décrite dans la section 2.9 page 27. Ces liaisons composites permettent de tenir compte du fait, qu'au cours du temps, un joueur a pu appartenir à plusieurs équipes, tout en s'assurant qu'au cours d'un match, un joueur ne peut jouer que dans l'équipe à laquelle il appartient alors. Ce schéma relationnel permet aussi de s'assurer qu'un but ne peut être marqué que par un joueur qui participe au match, en utilisant seulement l'intégrité référentielle (pas besoin de déclencheur). Il en va de même pour les cartons, en remarquant tout de même que pour les cartons et les buts, nous utilisons la minute à laquelle ces événements se produisent comme sous-numérotation (notion pour laquelle nous renvoyons le lecteur à la section 2.10 page 28).

Mais ce qui nous intéresse plus particulièrement, sur ce schéma, est la colonne `sur le terrain` de type `Oui/Non`, que nous avons ajoutée à la table `joueurs` afin de gérer très simplement les listes duales. Cette colonne `sur le terrain` vaut `Oui` lorsque le joueur joue actuellement sur le terrain et `Non` s'il est sur le banc de touche (en tant que remplaçant, remplacé, blessé ou exclu du jeu).

Grâce à cette colonne, la requête-contenu de chacune des deux listes duales est très simple (tableau 60, pour la liste des joueurs[49]).

les tables et leurs liaisons	`joueurs`
les colonnes et leur affichage	`numéro joueur`, `nom`, `sur le terrain` (non affiché)
les critères sur les lignes	`[sur le terrain] = Oui`
les tris et leur sens	aucun

TAB. 60 – *Requête-contenu pour un liste duale*

49. nous en déduisons facilement celle pour la liste des remplaçants

La programmation du bouton `BoutonVersDroite` >, consiste simplement à positionner à `Oui` cette colonne `sur le terrain`, puis à mettre à jour les deux listes duales :

```
Private Sub BoutonVersDroite_Click()

  Dim NumeroItem As Variant

  For Each NumeroItem In RemplacantsDomicile.ItemsSelected

    DoCmd.RunSQL ("UPDATE joueurs SET [sur le terrain] = Yes" & _
    " WHERE [numéro joueur] = " & RemplacantsDomicile.ItemData(NumeroItem))

  Next NumeroItem

  RemplacantsDomicile.Requery
  JoueursDomicile.Requery

End Sub
```

La programmation du bouton symétrique (vers la gauche) est elle même obtenu par symétrie de la procédure précédente :

```
Private Sub BoutonVersGauche_Click()

  Dim NumeroItem As Variant

  For Each NumeroItem In JoueursDomicile.ItemsSelected

    DoCmd.RunSQL ("UPDATE joueurs SET [sur le terrain] = No" & _
    " WHERE [numéro joueur] = " & JoueursDomicile.ItemData(NumeroItem))

  Next NumeroItem

  RemplacantsDomicile.Requery
  JoueursDomicile.Requery

End Sub
```

Notons qu'il s'agit de la première fois que nous utilisons `DoCmd.RunSQL` avec une requête de mise à jour (la syntaxe de l'instruction `UPDATE` est décrite dans la section 4.3 page 57). En effet, les `DoCmd.RunSQL` de la section précédente n'a fait appel qu'à des instructions `INSERT` ou `DELETE`.

Par ailleurs, nous pouvons copier et coller le contenu de ces deux procédures dans les événements `Sur double clic` des deux listes duales, afin que l'utilisateur ait le choix entre cliquer sur les boutons ou double-cliquer sur les listes.

Il reste à gérer automatiquement l'exclusion d'un joueur sur carton rouge. Pour cela nous réagissons à deux événements :

- d'abord, à l'événement `Après MAJ` (autrement dit, sur mise à jour) de la case à cocher `CartonRouge`, afin de mettre à jour la colonne `sur le terrain` ;
- ensuite, à l'événement `Sur sortie` du sous-formulaire des cartons, afin de mettre à jour le contenu des listes duales :

```
Private Sub CartonRouge_AfterUpdate()

  If(CartonRouge.Value = True) Then

    DoCmd.RunSQL ("UPDATE joueurs SET [sur le terrain] = No " & _
    "WHERE [numéro joueur] = " & NumeroJoueur.Value)

  Else

    DoCmd.RunSQL ("UPDATE joueurs SET [sur le terrain] = Yes " & _
    "WHERE [numéro joueur] = " & NumeroJoueur.Value)

  End If

End Sub

Private Sub CartonsDomicile_Exit(Cancel As Integer)

  RemplacantsDomicile.Requery
  JoueursDomicile.Requery

End Sub
```

Notons que cette solution, fondée sur l'ajout d'une colonne `Oui/Non` supplémentaire dans la table qui sert de contenu aux listes duales, ne fonctionne que si un seul formulaire ayant besoin de cette colonne n'est ouvert à la fois à un instant donné, sans quoi il pourrait y avoir des conflits entre les formulaires. Il s'agit d'une hypothèse contraignante, mais qui permet de gérer très simplement les listes duales.

8.6 Ajouter une valeur manquante dans une liste déroulante

Dans la liste déroulante des nationalités sur le formulaire de la figure 1(b) page 1, il peut manquer à l'utilisateur sa propre nationalité. La table qui correspond à cette liste déroulante est la table de codification `nationalités` sur le schéma relationnel de la figure 4 page 7. Comme il s'agit d'une table réduite à une colonne (hormis sa clé étrangère en numérotation automatique), il est possible de proposer à l'utilisateur d'ajouter directement sa nationalité dans la liste, sans en passer par un formulaire de saisie (qui serait ridiculement simpliste, d'ailleurs).

Pour cela, il faut que l'utilisateur saisissent lui-même sa nationalité dans la liste déroulante. La liste détecte alors automatiquement l'absence de cette valeur parmi les nationalités déjà présentes. Nous pouvons alors réagir à l'événement `Sur absence dans liste` (`NotInList`), à la fois pour éviter le message d'erreur soulevé par le fait que la propriété `Limiter à liste` est positionnée à `Oui` (conformément au conseil formulé dans la section 5.3.6 page 74) et pour proposer à l'utilisateur d'enregistrer sa valeur (avec un message de confirmation comme sur la figure 127, au cas où il s'agit d'une erreur de saisie).

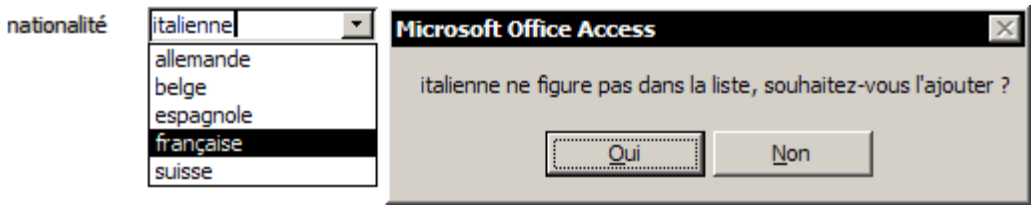


FIG. 127 – *Ajout d'une nationalité dans la table de modification correspondante*

La programmation de la procédure correspondante repose simplement sur l'exécution d'une requête d'insertion *via* une commande `DoCmd.RunSQL` :

```
Private Sub ListeNationalites_NotInList(NewData As String, Response As Integer)

Dim Reponse As Integer

Reponse = MsgBox(NewData & " ne figure pas dans la liste, souhaitez-vous l'ajouter ?",
                   vbYesNo)

If Reponse = vbYes Then

  DoCmd.RunSQL ("INSERT INTO nationalités (nationalité) VALUES (""" & NewData & """)")
  Response = acDataErrAdded

 Else

  Response = acDataErrContinue

End If

End Sub
```

La colonne `numéro nationalité` de la table `nationalités` ne doit pas figurer dans la clause `INSERT INTO` car elle est en numérotation automatique. Par ailleurs, l'affectation de l'argument `Response` par les constantes VBA `acDataErrAdded` ou `acDataErrContinue` permet d'éviter l'affichage d'un message d'erreur à l'utilisateur.

9 Application autonome

Lorsque les futurs utilisateurs d'une base de données Access ne sont pas les mêmes personnes que le développeur de la base, celui-ci doit finaliser l'interface de cette base et gérer son déploiement.

9.1 Menu général

Afin que l'utilisateur final n'ai pas l'impression d'utiliser Access, mais une application VBA spécifique, nous pouvons créer un formulaire d'accueil :

- qui s'affiche automatiquement au démarrage de la base de données ;
- qui regroupe les formulaires et les états utiles ;
- et qui masque le reste.

Ce menu général est créé à l'aide de l'assistant prévu pour (figure 128), qui est disponible *via* le menu `Outils`, `Utilitaires de base de données`, `Gestionnaire de Menu Général`.

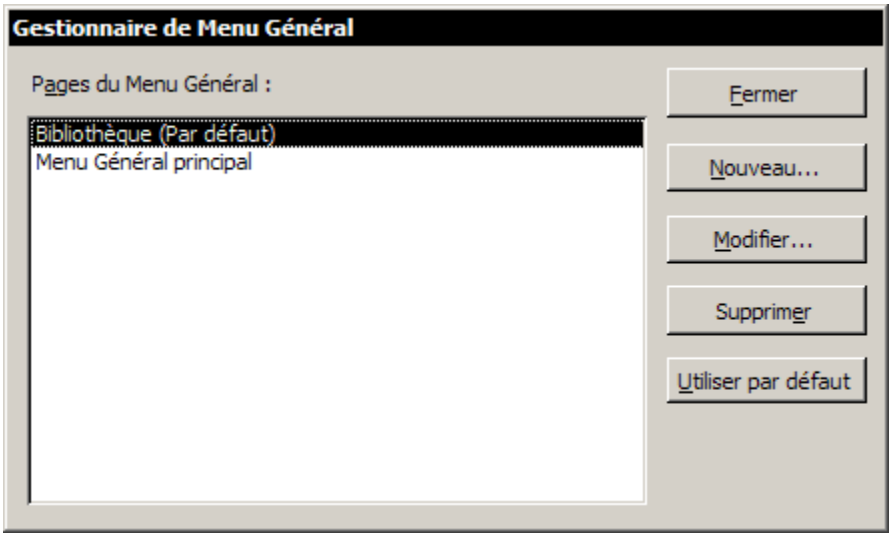


Fig. 128 – *Gestionnaire de menu général*

Dans cette boîte de dialogue, nous pouvons ajouter des pages à l'aide du bouton `Nouveau...`. Pour éditer une page (la page `Menu Général principal` par exemple), il suffit de la sélectionner puis de cliquer sur le bouton `Modifier...`.

Apparaît alors la fenêtre de la figure 129 qui permet d'ajouter (bouton `Nouveau...`) de nouveaux éléments à la page ou de modifier les éléments déjà présents.

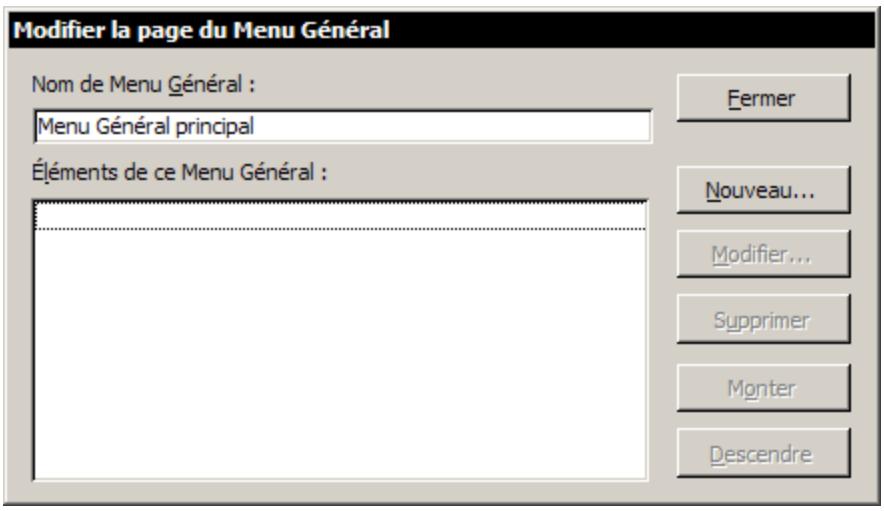


FIG. 129 – *Modification d'une page du menu général*

Dans une page du menu général, nous pouvons créer des boutons qui permettent d'effectuer une action parmi la liste proposée sur la figure 130.

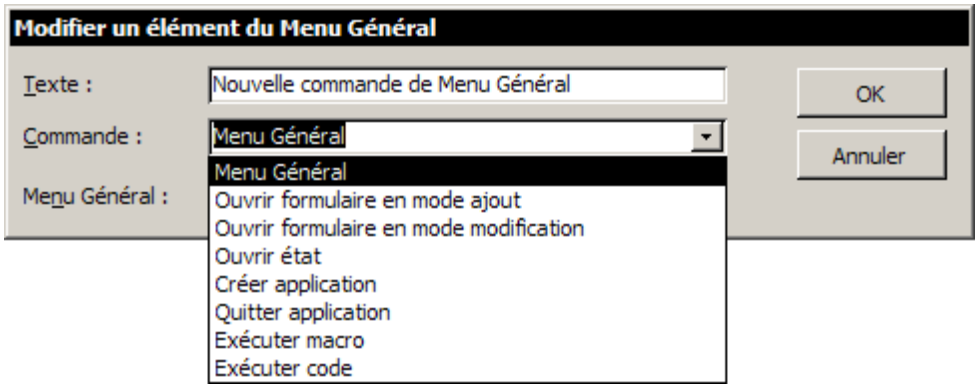


FIG. 130 – *Commandes possibles dans une page du menu général*

Parmi les commandes possibles, nous distinguons `Ouvrir formulaire en mode ajout` qui convient aux formulaires de saisie et `Ouvrir formulaire en mode modification` qui convient aux formulaires de modification, ainsi qu'aux formulaires de consultation.

Par exemple, dans une base de données pour une bibliothèque, les formulaires intéressants et les états intéressants pour l'utilisateur peuvent être accessibles *via* le formulaire principal de la figure 131.

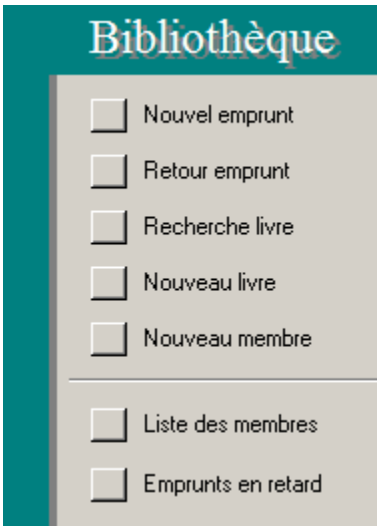


FIG. 131 – *Formulaire principal pour une base de données dédiée à une bibliothèque*

Une fois le formulaire `Menu Général` créé, il reste à l'afficher automatiquement au démarrage de l'application (à l'ouverture du fichier). Pour cela, il faut ouvrir la boîte de dialogue de la figure 132 avec le menu `Outils`, `Démarrage...`.

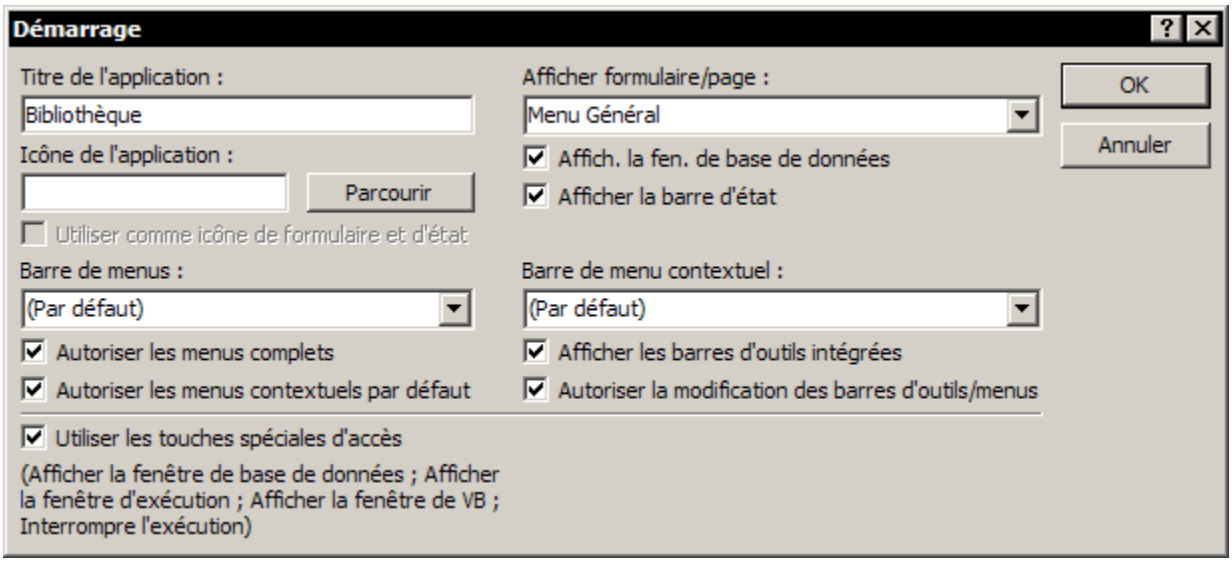


FIG. 132 – *Affichage du menu général au démarrage*

Le formulaire principal d'une base de données est un formulaire modal[50] , autrement dit, il exige une réponse avant de pouvoir continuer (comme une boîte de dialogue).

Il est également conseillé d'ajouter une barre d'outils avec des menus personnalisés pour accéder à ces mêmes formulaires et états à tout moment dans l'application (section suivante).

50. il ne faut pas confondre les formulaires modaux avec les formulaires indépendants, qui sont ceux qui restent affichés au premier plan, même si l'utilisateur travaille dans un autre formulaire ; le caractère modal et le caractère indépendant d'un formulaire, peuvent être modifiés dans l'onglet `Autres` de la fenêtre des propriétés du formulaire (figure 63 page 62)

9.2 Menus personnalisés

Pour créer un menu spécifique pour une application comme sur la figure 133, la procédure est classique pour un logiciel Office, il suffit d'ouvrir le menu `Outils`, `Personnaliser...`.

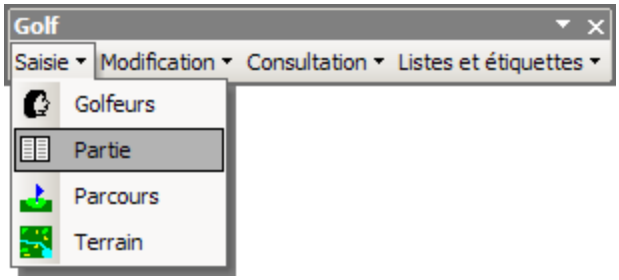


FIG. 133 – *Menu personnalisé pour l'application* `Golf`

Dans l'onglet `Barres d'outils` (figure 134), nous pouvons créer une nouvelle barre d'outils avec le bouton `Nouvelle...`.

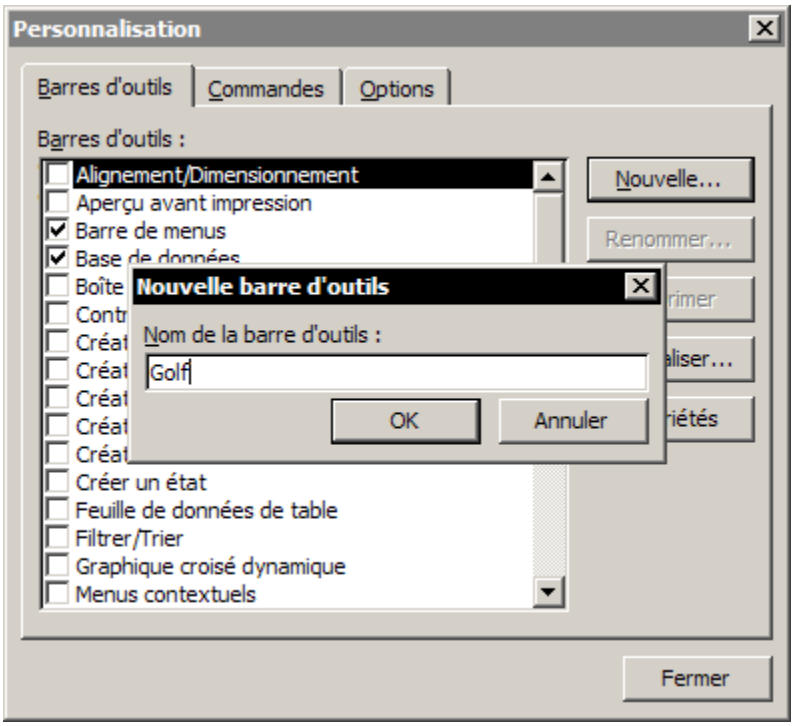


FIG. 134 – *Création d'une nouvelle barre intitulée* `Golf`

Dans l'onglet `Commandes` (figure 135), grâce à la catégorie `Nouveau menu`, nous pouvons ajouter, à la souris, autant de menus que nécessaire.

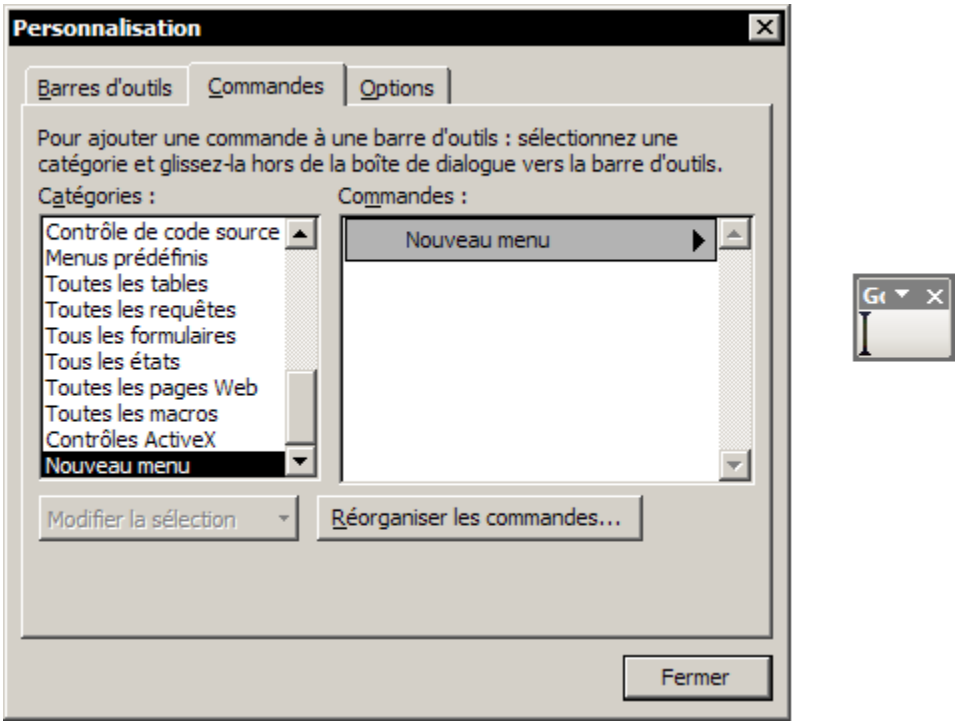


FIG. 135 – *Ajout de menu dans la barre personnalisé*

Pour renommer un nouveau menu, il suffit de cliquer droit dessus et de saisir le nom désiré dans la zone prévue pour (figure 136).

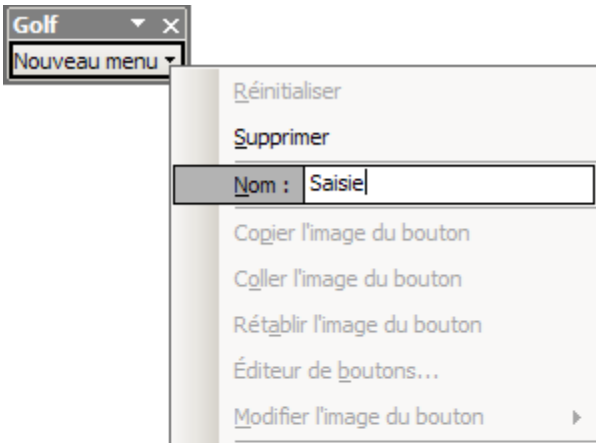


FIG. 136 – *Menu contextuel pour renommer un nouveau menu*

Nous disposons ensuite des catégories `Tous les formulaires` et `Tous les états`, pour ajouter une commande qui ouvre un formulaire ou un état (figure 137).

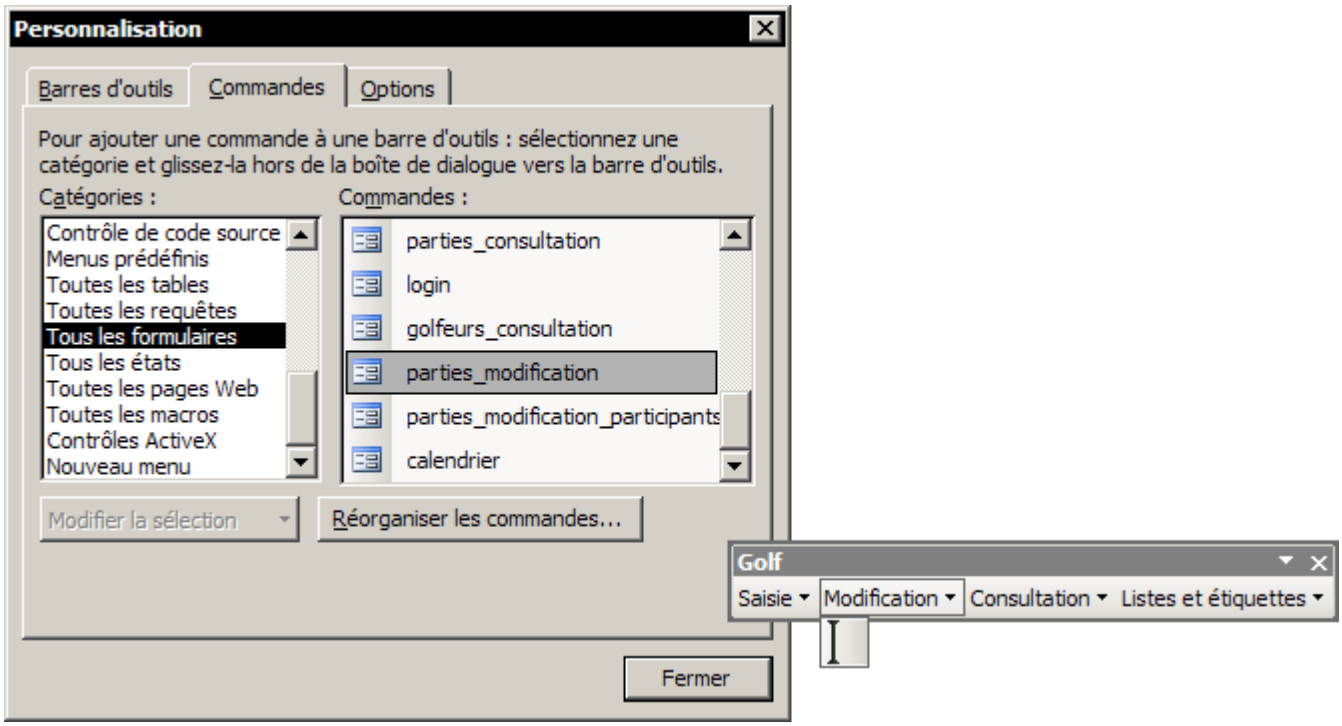


FIG. 137 – *Ajout d'une commande à l'un des menus*

Nous laissons au lecteur le soin de trouver le menu contextuel qui permet d'éditer l'icône d'une commande (figure 138).

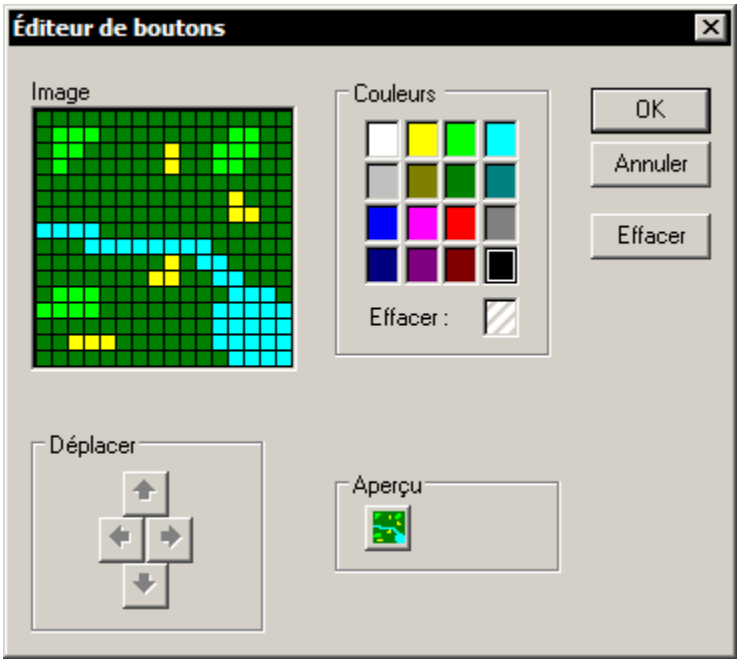


FIG. 138 – *Édition, pixel par pixel, de l'image d'une commande*

L'ensemble de cette procédure permet de rendre facilement accessibles les principaux formulaires et les principaux états de l'application Access.

9.3 Identification

Access fournit des services de sécurité, disponibles *via* le menu `Outils`, `Sécurité`, qui permettent de définir des utilisateurs, des groupes d'utilisateurs et de leur attribuer des droits[51]. Malheureusement, sans déploiement adéquat, cette stratégie de sécurité est attachée à l'instance d'Access dans laquelle elle est instaurée, au lieu d'être attaché à un fichier Access qui peut être ouvert sur plusieurs postes distincts. Il s'agit donc plutôt d'une sécurité de SGBD, que d'une sécurité orientée bureautique.

Il est par ailleurs possible de définir un mot de passe global sur un fichier Access, mais cela ne permet pas de définir différents rôles spécifiques pour les utilisateurs de cette base. Nous proposons, dans cette section, l'élaboration d'un formulaire de connexion rudimentaire qui peut ensuite être utilisé pour différencier les utilisateurs connectés.

Nous prenons l'exemple d'une application contenant des données sensibles ou personnalisées, à l'ouverture de laquelle l'utilisateur est invité à s'identifier à l'aide du formulaire de la figure 139).

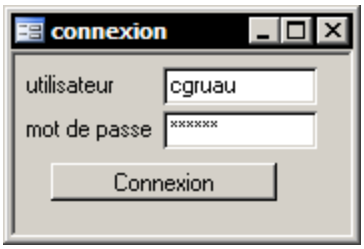


FIG. 139 – *Formulaire de connexion*

Ce formulaire repose sur l'existence d'une table qui regroupe les utilisateurs de la base de données (figure 140).

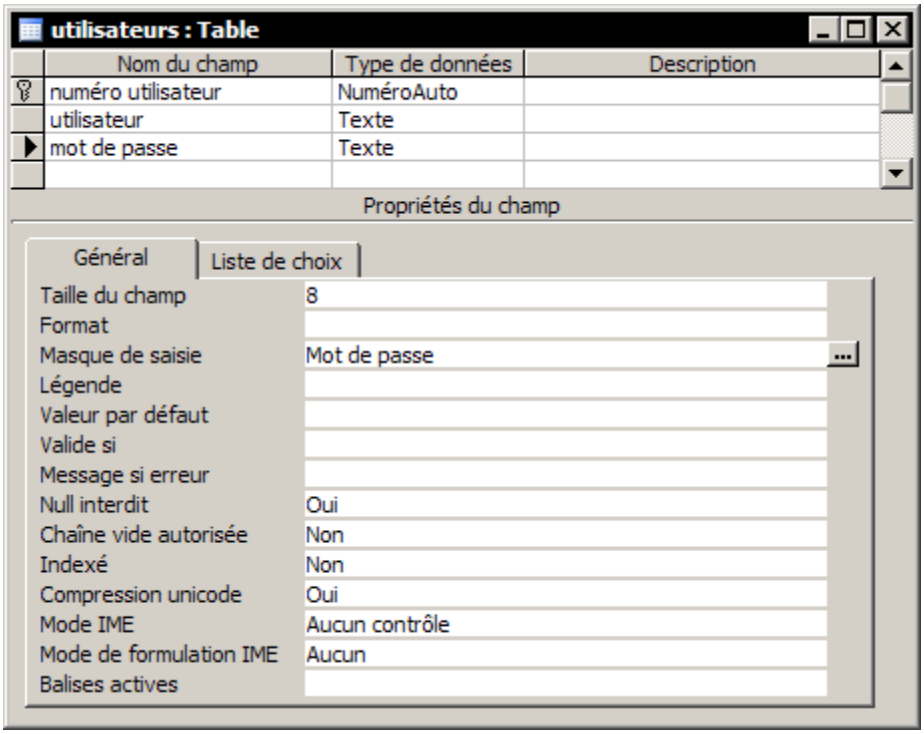


FIG. 140 – *Table des utilisateurs pour le formulaire de connexion*

51. pour une description détaillée de ces fonctionnalités, nous renvoyons le lecteur vers les références [self-access, ybet]

Cette table comporte trois colonnes :
- une clé primaire en numérotation automatique (ce devrait désormais être un réflexe) ;
- une colonne simple, de type `Texte` pour le nom d'utilisateur qu'il est conseillé d'indexer sans doublons (comme sur la figure 22 page 23), afin de ne pas avoir deux fois le même utilisateur ;
- une troisième colonne de type `Texte` munie du masque de saisie `*******` (comme sur la figure 7 page 10) pour le mot de passe.

Les mots de passe n'apparaissent alors jamais clairement, y compris lorsqu'il sont saisis directement dans la table `utilisateurs` en mode feuille de données (figure 141).

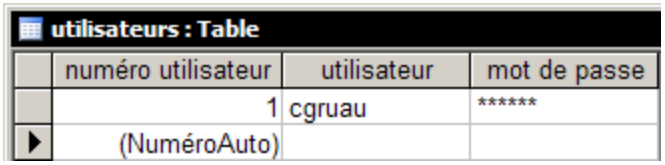


FIG. 141 – *Saisie d'un nouvel utilisateur et mot de passe masqué*

Il est nécessaire de protéger l'accès à cette table, sans quoi n'importe quel utilisateur peut en modifier le contenu. Notamment, nous pouvons masquer l'existence de la table `utilisateurs` en éditant ses propriétés (clic droit sur la table dans la liste des tables) puis en cochant la case `Masqué` (figure 142).

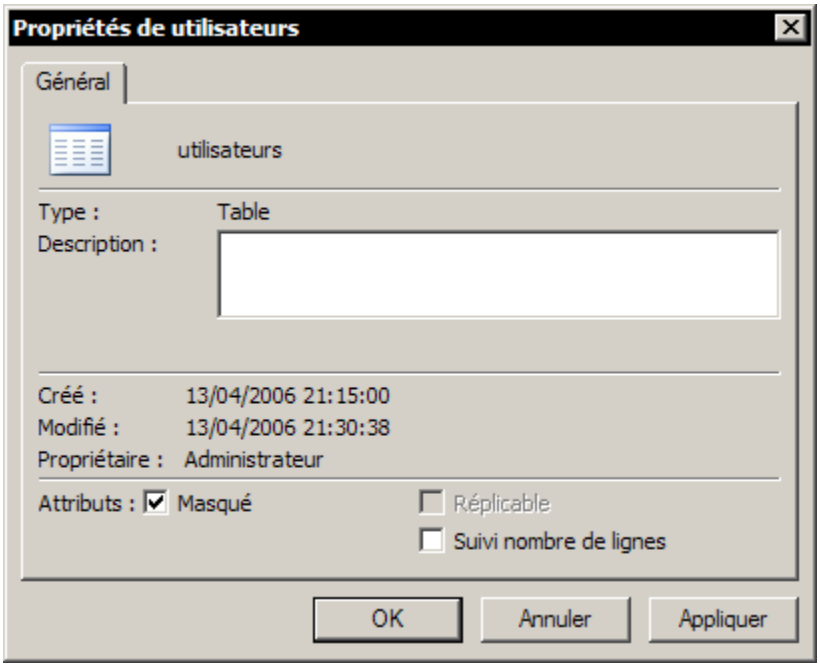
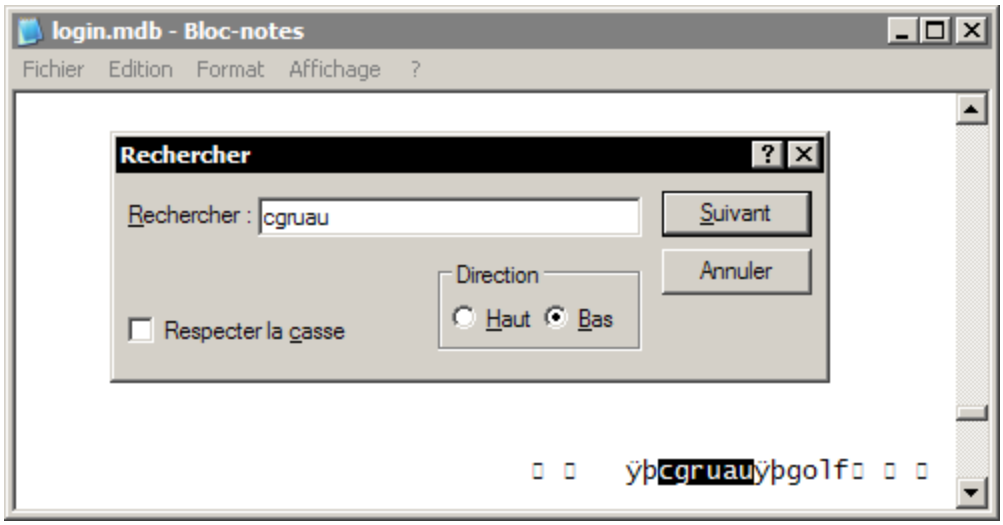
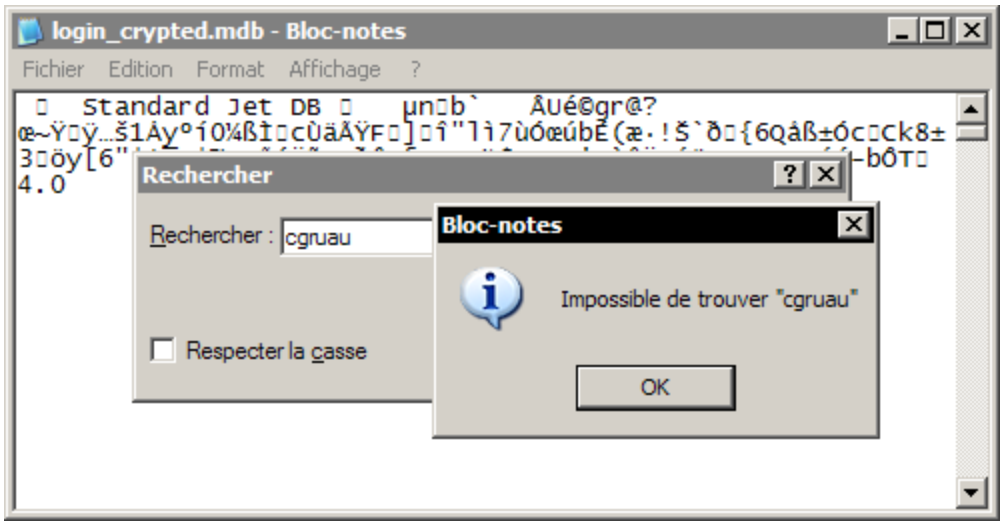


FIG. 142 – *Rendre invisible une table dans la liste des tables*

Il est par ailleurs indispensable de crypter la base de données (menu `Outils`, `Sécurité`, `Coder une base de données`), sans quoi les mots de passe de la table `utilisateurs` peuvent apparaître en clair, si le fichier `.mdb` de la base est lu avec un éditeur de texte (figure 143).



(a) sans cryptage



(b) avec cryptage

FIG. 143 – *Édition d'un fichier Access à l'aide d'un éditeur de textee, à la recherche d'un mot de passe*

Maintenant, la programmation du bouton `Connexion` de la figure 139 peut être la suivante :

```
1  Private Sub Connexion_Click()
2  On Error GoTo Reinitialisation
3
4    Dim Nombre As Integer
5
6    Nombre = DCount("utilisateur", _  ' la colonne utilisateur
7                    "utilisateurs", _ ' de la table utilisateurs
8                    "utilisateur = """ & Login.Value & """" & _
9                    "AND [mot de passe] = """ & Password.Value & """")
10
11   If Nombre <= 0 Then
12
13     MsgBox("connexion refusée")
14
15 :Reinitialisation
16
17     Login.Value = ""
18     Password.Value = ""
19     Login.SetFocus
20
21   Else
22
23     DoCmd.OpenForm "Menu Général ", , , ""
24     DoCmd.Close acForm, "connexion"
25
26   End If
27
28 End Sub
```

Cette procédure compte le nombre d'utilisateurs qui ont le même nom d'utilisateur et le même mot de passe que celui qui ont été saisis dans les contrôles `Login` et `Password` (normalement, le résultat est 0 ou 1). Si le résultat est nul (ou négatif, par malheur), la connexion est refusée, les contrôles sont vidés et l'utilisateur est invité à saisir un nouvel utilisateur. Sinon, le formulaire du menu général est ouvert et le formulaire de connexion est fermé.

Pour afficher le caractère « double quote » `"` dans une chaîne de caractères (délimitée par deux `"`, rappelons-le), il suffit de doubler le caractère `"`.

La série de trois caractères `"""` de la ligne 7 s'interprètent donc comme le caractère `"`, doublé pour figurer dans la chaîne, qui est close par le troisième caractère `"`. La série de quatre caractères `""""` de la ligne 7 commencent par ouvrir la chaîne (premier `"`) et à placer une double quote (les deux suivants) à l'intérieur de la chaîne, puis à clore cette chaîne (quatrième `"`). Cela permet d'encadrer la valeur du contrôle `Login` par un `"` au début et un `"` à la fin, ce dont a besoin la syntaxe SQL.

Il en va de même pour encadrer la valeur du contrôle `Password`.

Enfin, le formulaire `Menu Général` est celui qui a été créé par Access dans la section 9.1 page 139. Malheureusement, le dernier caractère de la chaîne `"Menu Général "` est un caractère spécial (pour une raison mystérieuse). Il faut donc copier-coller cette chaîne depuis un autre bouton qui ouvre ce formulaire `Menu Général`, à condition que ce bouton soit créé automatiquement par l'assistant Access prévu à cet effet.

9.4 Fractionnement et compilation

Quand une première mouture de notre application est achevée, il est recommandé de fractionner le fichier `.mdb` d'une base de données Access en deux fichiers :

- un fichier `donnees.mdb` contenant les tables de la base et leurs clés étrangères ;
- et un fichier `application.mdb` contenant les requêtes, les formulaires et les états de la base.

Toutes les tables indispensables du fichier `donnees.mdb` doivent alors être attachées dans la base de données `application.mdb` (avec le menu `Fichier`, `Données externes`, `Lier les tables...`), afin que les requêtes, les formulaires et les états ne soient pas orphelins.

Les avantages d'un tel fractionnement sont multiples :

- il peut y avoir plusieurs fichiers spécifiques `application1.mdb`, `application2.mdb`, etc. qui utilisent un même fichier `donnees.mdb`, ce qui est plus clair pour les utilisateurs qui ont besoin d'une application spécifique ;
- le développeur peut modifier le fichier `application.mdb` sans se soucier des autres fichiers ;
- l'administrateur peut sauvegarder le fichier `donnees.mdb` sur un autre disque sans se soucier des autres fichiers ;
- enfin, le fichier `application.mdb` peut être compilé en un fichier `application.mde`, dans lequel l'exécution du code VBA est plus rapide et pour lequel les formulaires et les états ne sont pas modifiables par l'utilisateur (ce qui est parfois une bonne chose).

La compilation du fichier `application.mdb` en un fichier `application.mde` (menu `Outils`, `Utilitaires de base de données`, `Créer un fichier MDE...`), exige :

- la compilation[52] préalable du fichier `donnees.mdb` en un fichier `donnees.mde` ;
- et l'attache des tables indispensables du fichier `donnees.mde` dans la base de données `application.mde` ;

car seules les tables d'un fichier `.mde` peuvent être attachées dans un fichier `.mde`.

Le développeur continue de travailler sur les fichiers `.mdb` (qu'il faut donc conserver) et ne fournit aux utilisateurs que les fichiers `.mde` correspondants. Notons que pour ouvrir un fichier `.mde`, Access doit quand même être installé sur le poste de l'utilisateur, à moins d'ajouter le runtine Access au fichier `.mde` avec MS Office Developer.

Le lecteur désireux de connaître tous les détails concernant le fractionnement, la compilation et le déploiement, peut les trouver dans le document complet [Ambrosino], ainsi que dans l'article [Kallal], plus léger.

52. cette compilation consiste simplement en une recopie, puisque `donnees.mdb` ne contient pas de code VBA.

Conclusion

De nombreuses notions ne sont pas approfondies dans ce document. Notamment, les pages web, la dynamique croisée des tableaux et des graphiques, la collaboration d'Access avec d'autres logiciels, qui sont des fonctionnalités très importantes mais qui ne font pas partie des chapitres abordés ici[53].

Mais le but de ce support de formation est de comprendre que, fondamentalement, Access est à la fois :

- un outil RAD, c'est-à-dire Rapid Application Development, pour la bureautique (ce sont les formulaires) ;
- qui embarque, pour des raisons d'autonomie, un système de gestion de bases de données locales (ce sont les tables, leurs clés et les requêtes) ;
- et un outil de reporting basique (ces sont les états) ;

et de montrer qu'il est possible d'opter pour un travail rigoureux et efficace, à chaque étape.

Malheureusement, Access est voué à un usage personnel ou professionnel restreint, car :

- la taille d'une base est limitée à 2 Go ;
- au-delà de quelques connexions simultanées, les performances chutent et les interactions entre l'utilisateur et sa base de données sont lentes.

Au final, c'est un très bon outil pour de petites applications ou pour mettre en place des prototypes et des démonstrations auprès des utilisateurs finaux (conformément au constat dressé par [Pailleret]). Au delà, il est nécessaire d'investir dans une solution logicielle plus élaborée, comprenant notamment un couple associant un SGBDR et une plate-forme de développement objet, comme par exemple : SQL Server et Visual Studio, Oracle et J2EE ou encore un outil MySQL/PhP.

Cependant, les conseils généraux énoncés dans ce document, concernant la conception du schéma relationnel, l'ergonomie des formulaires, la création des états ou la programmation VBA - SQL, restent valables avec toutes ces solutions logicielles.

Remerciements

L'auteur tient à remercier chaleureusement Vincent Isoz, Fabrice Fournier et Etienne Pailleret pour leur expertise et pour leur travail de relecture sur ce document.

53. les références de la page suivante constituent de bons compléments sur ces sujets

Références

[Ambrosino] Ambrosino, J.-P. *Déploiement et empaquetage d'applications professionnelles Microsoft Access 2003*. http://argyronet.developpez.com/office/access/runtime/2003/

Ce document décrit de manière suffisamment détaillée les étapes de finalisation d'une application Access autonome et distribuable à l'aide du Runtime.

[Brouard et Soutou] Brouard, F. et Soutou, C. *SQL*. Pearson Education (Synthex), ISBN 2-7440-7095-5, 2005.

Ce livre contient notamment de nombreux exemples de construction de requête en SQL.

[Gruau] Gruau, C. *Conception d'une base de données*. http://cyril-gruau.developpez.com/merise/

La lecture de ce document est fortement conseillée pour comprendre les étapes normalisées de construction d'un schéma relationnel.

[Haught] Haught, D. *Making Access faster - 168 tips to speed up your apps!*. http://www.fmsinc.com/tpapers/faster/index.html

Cet page anglophone regroupe de nombreux conseils pour optimiser l'interactivité entre l'utilisateur et sa base Access, y compris en réseau.

[Isoz] Isoz, V. *MS Access*. http://sciences.ows.ch/infolearn/msaccess.pdf

Ce support de cours contient de nombreuses informations sur les possibilités du logiciel.

[Kallal] Kallal, A. *Splitting a access database, or how to run ms-access in a multi-user mode*. http://www.members.shaw.ca/AlbertKallal/Articles/split/index.htm

Cet article anglophone donne quelques détails concernant l'installation d'une application Access sur plusieurs postes utilisateurs.

[Pailleret] Pailleret, E. *Développement professionnel avec Access*. http://access.developpez.com/accessdevpro/

Un bilan sans concession de l'utilisation d'Access dans le milieu professionnel.

[self-access] *Sécurité Access*. www.self-access.com/access/securite/acSecurite.htm

Ce tutoriel dédié à la sécurité suffit à comprendre les fonctionnalités offertes par Access.

[Tylor et Anderson] Tylor, A. et Anderson, V. *VBA pour Access 2003*. Eyrolles, ASIN 2212114656, 2004.

Ce livre constitue une bonne référence en matière de développement VBA pour Access.

[ybet] *Cours Access en ligne*. Document disponible sur le site de www.ybet.be

La page 15 de ce support, consacrée à la sécurité Access, est très instructive.

Table des figures

1 Remplacement d'un formulaire classique par un formulaire Access . . . 1
2 Fenêtre d'accueil du logiciel . . . 5
3 Tables en mode création et définition des colonnes . . . 6
4 Schéma relationnel pour un formulaire de renseignement . . . 7
5 Liste des sous-types à choisir pour le type `Numérique` . . . 8
6 Définition d'un format d'affichage . . . 9
7 Masques de saisie prédéfinis . . . 10
8 Contraintes de non vacuité . . . 12
9 Règle de validation de table . . . 13
10 Le générateur d'expression . . . 13
11 Le menu le plus important d'Access, car il permet d'éditer le schéma relationnel de la base 14
12 Création d'une clé étrangère sur le schéma relationnel . . . 17
13 Schéma relationnel entre les tables `avions`, `compagnies` et `types d'avions` . . . 17
14 Réglages conseillés pour l'intégrité référentielle (cette fenêtre est obtenue en double cliquant sur la liaison à éditer) . . . 17
15 Clé étrangère obligatoire (non vide) . . . 18
16 Table de jonction pour une liaison plusieurs-à-plusieurs . . . 19
17 Création d'une clé primaire sur deux colonnes pour une table de jonction . . . 19
18 Autre colonne dans une table de jonction . . . 20
19 Application de la deuxième forme normale à une table de jonction . . . 20
20 Intervention dans la fenêtre des index . . . 21
21 Table historique . . . 22
22 Interdiction des doublons pour une liaison un-à-un . . . 23
23 Liaison un-à-un . . . 23
24 Une clé étrangère qui ne référence pas une clé primaire (exemple théorique) . . . 24
25 Deux tables filles qui héritent de leur table mère . . . 25
26 Utilisation de l'héritage pour séparer les colonnes principales des colonnes complémentaires 25
27 Auto-liaison un-à-plusieurs . . . 26
28 Deux auto-liaisons et donc deux duplicata pour la table de départ . . . 26
29 Clé étrangère composite . . . 27
30 Création d'une clé étrangère composite sur le schéma relationnel . . . 27
31 Utilisation de clés primaires partiellement étrangères . . . 28
32 Table de jonction ternaire . . . 29
33 Les différents modes pour les requêtes . . . 30
34 Requête nommée `livres_empruntables` et utilisée par la suite . . . 31
35 La feuille des propriétés d'une requête . . . 32
36 Limitation de l'affichage du résultat aux premières lignes . . . 33
37 Requête impliquant plusieurs tables . . . 34
38 Vue isolée sur le schéma d'une nouvelle requête . . . 35
39 Rappel du schéma relationnel concernant les paris de courses de chevaux . . . 36
40 Critères écrits sur plusieurs rangées . . . 39
41 Mode `Totaux` et opérations . . . 40
42 Rappel de l'utilisation de l'héritage pour séparer les colonnes principales des colonnes complémentaires . . . 45
43 Choix de la bonne jointure externe . . . 45
44 Utilisation du générateur d'expression pour saisir la formule d'une colonne ou d'un critère, à partir des fonctions Access . . . 46
45 Jointure externe avec une table additionnelle côté facultatif . . . 47
46 Utilisation d'auto-jointures externes afin de préserver toutes les lignes du résultat . . . 48
47 Présence d'un cycle sur le schéma relationnel global d'une base de données . . . 49
48 Les différents types de requête . . . 51

49 Cases à décocher pour éviter les messages de confirmation à l'exécution d'une requête d'action . . . 51
50 Table destinée à être remplie par une requête d'insertion . . . 52
51 Boîte de dialogue qui apparaît lors de la conversion d'une requête de sélection en une requête d'insertion . . . 52
52 Requête d'action de type ajout . . . 53
53 Schéma relationnel pour la table des livres non empruntables . . . 54
54 Propriétés d'une requête d'insertion . . . 55
55 Requête d'action de type suppression . . . 56
56 Requête d'action de type mise à jour . . . 57
57 Rappel des réglages conseillés pour l'intégrité référentielle (cette fenêtre est obtenue en double cliquant sur la liaison voulue dans la fenêtre du schéma relationnel) . . . 58
58 Nouvelle colonne qu'il faudra remplir à l'aide d'un regroupement . . . 59
59 Boîte de dialogue qui apparaît lors de la conversion d'une requête de sélection en une requête de création de table . . . 59
60 Propriétés d'une requête de création de table . . . 60
61 Formulaire papier . . . 61
62 Formulaire informatique avec interface graphique . . . 61
63 Principales propriétés à régler à la création d'un formulaire (onglet `Format`) . . . 62
64 Principales propriétés à régler à la création d'un formulaire (onglet `Données`) . . . 63
65 Les différents modes possibles pour un formulaire . . . 64
66 Composition d'un formulaire de saisie . . . 64
67 Composition d'un formulaire en mode continu . . . 65
68 Assistant de création d'un nouveau groupe d'options . . . 68
69 Réglage de la source contrôle d'un groupe d'options sur une clé étrangère . . . 69
70 Principales propriétés à régler à la création d'une liste . . . 70
71 Assistant de création d'une nouvelle liste déroulante . . . 71
72 Liste déroulée, avec ses intitulés de colonnes . . . 72
73 Stockage des valeurs pour une toupie par zone de liste . . . 72
74 Choix du mode de sélection multiple . . . 73
75 Listes duales avec boutons de transfert . . . 74
76 Utilisation d'un sous-formulaire . . . 75
77 Choix du mode sous-formulaire dans l'assistant de création d'un formulaire . . . 76
78 Deux façon de gérer les tabulations . . . 78
79 Verrouillage et désactivation d'un contrôle . . . 79
80 Boîte de dialogue de la recherche parmi les fiches d'un formulaire . . . 80
81 Filtre défini à travers un formulaire de saisie converti en formulaire de consultation . . . 80
82 Les différents modes d'affichage des objets Access . . . 82
83 Choix de la mise en page (c'est crucial pour un état) . . . 83
84 Description des différentes zones d'un état, sur une composition manuelle . . . 84
85 Création d'un état avec l'assistant . . . 86
86 État généré par l'assistant avant correction . . . 87
87 Le même état, mais avec une disposition améliorée . . . 88
88 L'état corrigé en mode affichage est prêt à être imprimé et diffusé . . . 89
89 Formulaire de modification des groupes et du sens des tris dans un formulaire . . . 90
90 Composition d'un annuaire (échantillon) . . . 90
91 Tables utilisées pour établir l'annuaire des employés . . . 91
92 Création d'un annuaire avec l'assistant . . . 92
93 Annuaire généré par l'assistant avant correction . . . 93
94 Le même annuaire, mais mieux organisé . . . 93
95 L'annuaire corrigé en mode affichage est prêt à être imprimé et diffusé . . . 94
96 Formulaire intégré pour une mise en forme conditionnelle . . . 94
97 Mise en forme conditionnelle basée sur une expression . . . 95

98 Utilisation de deux sous-états 96
99 Tables utilisées pour l'état comportant deux sous-états 96
100 Création de l'état principal avec l'assistant 97
101 Création du premier sous-état avec l'assistant 98
102 L'état principal complet, muni de ses deux sous-états dans la zone détail 99
103 Fenêtre Visual Basic for Applications avec, à gauche, la liste des modules et à droite, la feuille de programmation du module sélectionné 100
104 Boîte de dialogue `MsgBox` 103
105 Boîte de dialogue pour insérer une nouvelle routine dans le module 103
106 Explorateur d'objets VBA 109
107 Formulaire et sous-formulaire dont la clé étrangère doit être non vide 110
108 Onglet `Événement` de la fenêtre des propriétés du formulaire 111
109 Annuaire comportant une ligne grisée sur deux 112
110 Formulaire muni de boutons programmés en VBA et formulaire appelé 114
111 Assistant de création d'un bouton de commande 115
112 Formulaire calendrier généraliste 117
113 Position d'un point d'arrêt dans le code VBA 118
114 Fenêtre des variables locales 119
115 Fenêtre des espions 120
116 Fenêtre d'exécution 120
117 Consultation de la pyramide des âges d'un pays au cours d'une année 122
118 Boîte de dialogue demandant le paramètre de la requête quand le formulaire n'est pas ouvert 123
119 Source contrôle d'un contrôle calculé basé sur une agrégation de domaine 124
120 Sous-formulaire muni d'une sous-numérotation 126
121 Filtres prédéfinis en VBA pour la consultation des livres 127
122 Schéma relationnel pour stocker des cartons d'invitation, avec une liaison plusieurs-à-plusieurs gérée par une liste à sélection multiple 129
123 Formulaire de saisie d'un carton de mariage, muni d'une liste à sélection multiple 130
124 Formulaire de modification dans lequel la zone de liste a été remplacée par un sous-formulaire 132
125 Formulaire de saisie d'une feuille de match 134
126 Schéma relationnel du formulaire précédent 135
127 Ajout d'une nationalité dans la table de modification correspondante 138
128 Gestionnaire de menu général 139
129 Modification d'une page du menu général 140
130 Commandes possibles dans une page du menu général 140
131 Formulaire principal pour une base de données dédiée à une bibliothèque 141
132 Affichage du menu général au démarrage 141
133 Menu personnalisé pour l'application `Golf` 142
134 Création d'une nouvelle barre intitulée `Golf` 142
135 Ajout de menu dans la barre personnalisé 143
136 Menu contextuel pour renommer un nouveau menu 143
137 Ajout d'une commande à l'un des menus 144
138 Édition, pixel par pixel, de l'image d'une commande 144
139 Formulaire de connexion 145
140 Table des utilisateurs pour le formulaire de connexion 145
141 Saisie d'un nouvel utilisateur et mot de passe masqué 146
142 Rendre invisible une table dans la liste des tables 146
143 Édition d'un fichier Access à l'aide d'un éditeur de textee, à la recherche d'un mot de passe 147

Index

A

`A partir de` . . . 56
absence dans liste . . . 138
accent . . . 102
`acDataErrAdded` . . . 138
`acDataErrContinue` . . . 138
action événementielle . . . 79
ActiveX . . . 66
adresse . . . 101
affectation . . . 102
`AfterUpdate` . . . 110, 123
agrégation . . . 38, 41, 85
    VBA . . . 124
`AjDate` . . . 85, 128
ajout . . . 52
alias . . . 37, 40, 70
ambiguïté . . . 34
analyse . . . 42, 82
`And` . . . 106
annuaire . . . 82, 90
apostrophe . . . 101
argument . . . 104
arrêt tabulation . . . 78
articulation de critères . . . 38
assistant
    état . . . 86
    bouton de commande . . . 115
    formulaire . . . 76
    groupe d'options . . . 68
    liste déroulante . . . 71
astérisque . . . 32, 56
attachement . . . 149
auto-jointure . . . 44
    externe . . . 48
auto-liaison . . . 26, 44, 74

B

`BackColor` . . . 109
barre d'outils . . . 142
`BeforeUpdate` . . . 110
bloc d'informations plurielles . . . 96
boîte de dialogue . . . 103
booléen . . . 7
`Boolean` . . . 101
boucle infinie . . . 107, 121
bouton . . . 64
    bascule . . . 68, 127
    de sauvegarde . . . 130
    de transfert . . . 74
    magique . . . 98
    radio . . . 67
`Byte` . . . 101

C

calendrier . . . 66
`Call` . . . 104
caractères spéciaux . . . 9
cascade . . . 18, 58, 132
case à cocher . . . 66
casse . . . 39
chaîne
    de caractères . . . 7, 101
        comparaison . . . 100
    vide . . . 12
champ . . . 30
    taille . . . 8
`CheckBox` . . . 109
choix multiple . . . 67
`Choose` . . . 106
clé étrangère . . . 14, 16, 19, 53, 58
    composite . . . 27
    orpheline . . . 54, 56
    type . . . 7
clé primaire . . . 11, 17, 33, 58, 67
    composite . . . 19
    partiellement étrangère . . . 28
    type . . . 7
classe . . . 101, 109
    `Form` . . . 113
clause . . . 41
`Click` . . . 110
`Close` . . . 111, 114
code barre . . . 11
code postal . . . 11
coder une base . . . 147
collection . . . 113
colonne . . . 6
    calculée . . . 59
    calculable . . . 38
    facultative . . . 25
    liée . . . 70
`ComboBox` . . . 109
`CommandButton` . . . 109
`Comme` . . . 39, 97
commentaire . . . 101
`Compare` . . . 100
compilation . . . 149
complétion automatique . . . 101, 118
composition
    d'un état . . . 84

d'un formulaire . . . 64
`Compte` . . . 38
concaténation . . . 37, 125, 127, 131
condition
de jointure . . . 34
de sélection . . . 38, 83
confirmation d'action . . . 51
connecteur . . . 17
en forme de flèche . . . 46
connexion . . . 145
`Const` . . . 102
contenu . . . 70
recalculé . . . 123
contrôle . . . 64, 109
ActiveX . . . 66
calculé . . . 77, 124
catégorie . . . 65
dépendant . . . 66
contrainte . . . 12
de colonne . . . 12
de table . . . 13
`ControlSource` . . . 117
correction . . . 88
court-circuit relationnel . . . 36
création de table . . . 59
critère . . . 30, 38, 80, 125
absence . . . 32
crypter . . . 147
`Currency` . . . 101
cycle . . . 49

D

date . . . 22, 39
`Date` . . . 101
courante . . . 39
`Date()` . . . 85, 102
`Date/Heure` . . . 7
`DateAdd` . . . 128
`Datepart` . . . 104
`DblClick` . . . 110
`DCount` . . . 124, 148
déboguer . . . 118
`Debug.Print` . . . 120, 121
décimale . . . 9
déclaration . . . 102
déclencheur . . . 135
`DELETE` . . . 56, 133
démarrage . . . 141
dénombrement . . . 38, 40, 85
dénullification . . . 48, 91
dépendance fonctionnelle . . . 29
dépendances fonctionnelles . . . 14
`Dernier` . . . 42, 85
désactivation . . . 79
désordre . . . 32
détail (zone) . . . 64, 75, 84, 97
deux points . . . 37
deuxième forme normale . . . 20
dièse . . . 39
différent (opérateur) . . . 38
`Dim` . . . 102
`DISTINCT` . . . 33
division
entière . . . 37
par zéro . . . 121
`DMax` . . . 126
`Do While` . . . 107
`DoCmd` . . . 114, 129
domaine . . . 124
`Double` . . . 101
double clic . . . 136
double quote . . . 102, 131
doublon . . . 11, 21, 23, 33
`DSum` . . . 124
duplicata . . . 26, 44

E

écran . . . 61
élimination des cycles . . . 49
`Empty` . . . 106
en-tête
état . . . 84
de colonne . . . 72
de formulaire . . . 64, 75
groupe . . . 84
`Enabled` . . . 109
`Enter` . . . 110
entier . . . 7, 101
incrémenté automatiquement . . . 7
`Entre` . . . 38
ergonomie . . . 61, 74, 78, 79, 100
`Err` . . . 121
erreur
de logique . . . 118
de syntaxe . . . 101, 118
`Error` . . . 121
espace . . . 38, 101
de travail . . . 104
esperluette . . . 37, 102
espion . . . 120
état . . . 82
principal . . . 97
étiquetage . . . 82
étiquette . . . 65
événement . . . 79, 110
`Error` . . . 121

ex-aequo . . . 43
exécution pas à pas . . . 118
`Exit` . . . 110
`Exit Do` . . . 108
`Exit For` . . . 108
`Explicit` . . . 100
explorateur d'objets . . . 109
expression . . . 95

F

faute de frappe . . . 6, 15
fenêtre
  d'accueil . . . 5
  d'exécution . . . 120
feuille de programmation . . . 100
fiche de renseignements . . . 1
fichier MDE . . . 149
`Filter` . . . 127
`FilterOn` . . . 127
filtre . . . 39, 80, 127
  et tri avancé . . . 81
  par formulaire . . . 80
fonction . . . 104
  d'agrégation . . . 38
  intégrée . . . 46
`Font` . . . 109
`For` . . . 107
`For Each` . . . 113, 125, 130, 136
`ForeColor` . . . 109
format
  d'affichage . . . 9
  prédéfini . . . 9
formes normales . . . 15
`Forms` . . . 113
formulaire . . . 61
  calendrier . . . 117
  de facturation . . . 75
  éléments constitutifs . . . 64
  indépendant . . . 141
  informatique . . . 61
  modal . . . 141
  papier . . . 1, 61
  principal . . . 75
  requête . . . 77
formule . . . 37, 46, 60
fractionnement . . . 149
`Function` . . . 104

G

`Gauche` . . . 93
générateur d'expression . . . 13, 46
gestionnaire de menu général . . . 139
`GoTo` . . . 105, 108
`GoToRecord` . . . 131
graphique . . . 82
griser . . . 112
`GROUP BY` . . . 40
groupe . . . 90
  d'options . . . 6, 68
groupement . . . 40

H

`HAVING` . . . 41
héritage . . . 25
  multiple . . . 25
historique . . . 22
homonymie . . . 25

I

icône . . . 144
identification . . . 145
`If` . . . 105
`IIf` . . . 105
imbrication . . . 106
`In` . . . 39
incrémentation automatique . . . 11
indentation . . . 101
index . . . 23, 146
  composite . . . 21
  tabulation . . . 79
inégalité . . . 50
information
  bulle . . . 119
  express . . . 102
  globale . . . 43
initiale . . . 93
`INNER JOIN` . . . 34
`INSERT` . . . 53, 130
insertion . . . 52
  automatique . . . 74
intégrité référentielle . . . 17, 28, 56, 135
`Integer` . . . 101
interface graphique . . . 61
interpréteur VBA . . . 118
intervalle . . . 92
intitulé . . . 7, 36, 37, 40, 46, 70
invisible . . . 146
`IsNull` . . . 110
isolation . . . 35
`ItemData` . . . 113
`ItemsSelected` . . . 113

J

`JOIN` . . . 34
jointure . . . 18, 34, 36
  auto . . . 44

externe . . . 45
ambiguës . . . 47
plusieurs-à-plusieurs . . . 47
facultative . . . 45
obligatoire . . . 45
joker . . . 39

L

`Label` . . . 109
largeur . . . 78, 83
`LBound` . . . 107, 108
`Let` . . . 102
liaison . . . 17, 34
artificielle . . . 36
auto . . . 26, 44
composite . . . 27, 74, 135
manuelle . . . 44
plusieurs-à-plusieurs . . . 11, 19, 67, 96, 111, 129
plusieurs-à-plusieurs-à-plusieurs . . . 29, 74
un-à-plusieurs . . . 16, 19, 67
un-à-un . . . 23, 74
limitation
à liste . . . 74, 138
de l'affichage . . . 33
`ListBox` . . . 109, 113
liste
à sélection multiple . . . 129
déroulée . . . 72
déroulante . . . 67, 70, 138
de valeurs . . . 39
duale . . . 73, 134
`Locked` . . . 109
`Long` . . . 101
`Loop` . . . 107
`Loop Until` . . . 107

M

`Majuscule` . . . 93
masque de saisie . . . 10, 66, 146
masquer une table . . . 146
matrice . . . 102, 104
`Max` . . . 38
mdb . . . 149
mde . . . 149
`Me` . . . 117, 127
menu
contextuel . . . 102
général . . . 139
personnalisé . . . 142
Merise . . . 29
message d'erreur . . . 12
méthode . . . 109
méthodologie . . . 41, 81
`Min` . . . 38
mise à jour . . . 57, 123
mise à plat . . . 50
mise en forme conditionnelle . . . 94
mise en page . . . 83
mode
`Formulaire` . . . 64
`Simple` . . . 73
`Totaux` . . . 40, 58
`Étendu` . . . 73
`création` . . . 64
aperçu . . . 99
continu . . . 62, 65
création . . . 6, 30, 99
débogage . . . 119
SQL . . . 30
texte . . . 61
unique . . . 62
module VBA . . . 100, 101
modulo . . . 37
`Monétaire` . . . 7
mot de passe . . . 145
`Move` . . . 109
`Moyenne` . . . 38
`MsgBox` . . . 103, 105, 120

N

`Next` . . . 107
niladique . . . 104
niveau . . . 85
nom
complet . . . 123
de routine . . . 104
de variable . . . 101
nombre de décimale . . . 9
non vide . . . 110
normalisation . . . 15
`Not` . . . 106
`NotInList` . . . 138
`Null interdit` . . . 12, 18, 53, 58
`Numérique` . . . 7
numéro
de sécurité sociale . . . 11
de téléphone . . . 11
ISBN . . . 11
numérotation
arbitraire . . . 20
automatique . . . 53
sous . . . 28
`Nz` . . . 48, 91

O

objet . . . 109

DoCmd . . . 114
Err . . . 121
octet . . . 8, 101
onglet . . . 78
Données . . . 63, 70
Format . . . 62, 70
Ou . . . 81
Commandes . . . 143
Événement . . . 111
Open . . . 111
OpenForm . . . 116
OpenQuery . . . 115
opérateur
arithmétique . . . 37
de comparaison . . . 106
de comparaisons . . . 38
logique . . . 38, 106
point d'exclamation . . . 113
opération . . . 37
agrégation . . . 85
options
de regroupement . . . 92
de synthèse . . . 86
Or . . . 106
ORDER BY . . . 31
ordre
décroissant . . . 90
tabulation . . . 78
orientation . . . 83
Où . . . 41, 56, 85
Oui/Non . . . 7
outil
Relations . . . 14, 17, 26
requête . . . 30, 44

P

paramètre . . . 123
parenthèse . . . 106
passer à la ligne . . . 101, 131
pied
état . . . 84
formulaire . . . 64, 75
groupe . . . 84
plage de valeurs . . . 106
plaque d'immatriculation . . . 11
point d'arrêt . . . 118
pré-agrégation . . . 85
première Forme Normale . . . 15
premières valeurs . . . 33
Premier . . . 42, 85
Print . . . 112
priorité . . . 106
Private . . . 104, 110
procédure . . . 104
produit cartésien . . . 34
propriété . . . 109
contrôle . . . 65
de formulaire . . . 62
de requête . . . 32, 55, 60
Public . . . 104
puissance . . . 37

R

RAD . . . 150
rangée . . . 39
Ajouter à . . . 52
Mise à jour . . . 57
Opération . . . 40
Supprimer . . . 56
recherche . . . 80
RecordSource . . . 117
redondance . . . 15, 16
réel
à virgule fixe . . . 7, 101
à virgule flottante . . . 8, 101
référence . . . 101, 102
brisée . . . 17
règle de validation . . . 12
de table . . . 13
regroupement . . . 40, 85
relations . . . 14, 17, 26
reporting . . . 5, 150
requête
action . . . 115
contenu . . . 71
d'action . . . 51
d'insertion . . . 52
d'un formulaire . . . 77
de création de table . . . 59
de mise à jour . . . 57, 60
de sélection . . . 30
de suppression . . . 56
nom . . . 59
paramétrée . . . 122
propriétés . . . 32
source . . . 77, 85
Requery . . . 109, 123, 137
Resume . . . 121
routine . . . 103
RunSQL . . . 129, 131, 136

S

saisie . . . 61
saut de page . . . 78
schéma
entités-associations . . . 14, 29

relationnel . . . 7, 17
d'une requête . . . 34, 50
de la base . . . 34
sécurité . . . 145
`SELECT` . . . 31, 41
`Select Case` . . . 106
sélection multiple . . . 73, 125, 129
`Set` . . . 102
`SetFocus` . . . 109, 148
SGBD . . . 5
`Sheets` . . . 102
`Single` . . . 101
`SizeToFit` . . . 109
somme . . . 38, 85
source contrôle . . . 70, 77, 124
d'un cadre . . . 69
sous-état . . . 96
sous-formulaire . . . 75, 76, 126, 132
sous-groupe . . . 42
sous-numérotation . . . 28, 74, 126, 135
sous-requête . . . 37
SQL . . . 5, 30, 122
`Step` . . . 107
`String` . . . 101
structure
de décision . . . 105
structures
de répétition . . . 107
`Sub` . . . 104
`SubForm` . . . 109
suppression . . . 56
surlignage . . . 72

T

table . . . 6
additionnelle . . . 47
de codification . . . 6, 16, 69, 74
de destination . . . 52
de jonction . . . 19, 25, 75
entre trois tables . . . 29
requête d'insertion . . . 54
destination . . . 55
source . . . 55
tableau . . . 102
de bord . . . 82
tabulation . . . 78
taille . . . 150
champ . . . 8
`TextBox` . . . 109
`Texte` . . . 7
tiret bas . . . 101
titre . . . 64
`TOP` . . . 33
totaux . . . 65
toupie . . . 66, 72, 126
tri . . . 31, 38, 80, 90
absence . . . 32
troisième Forme Normale . . . 15
type
de données
Access . . . 7
VBA . . . 101
de formulaire . . . 62
de jointure . . . 45
de requête . . . 51

U

`UBound` . . . 107, 108
unicité . . . 21, 23
unité . . . 9
`UPDATE` . . . 57, 136

V

vacuité . . . 12
valeur
de renvoi . . . 104
distinctes . . . 33
manquante . . . 138
par défaut . . . 17, 67, 126
premières . . . 33
validée . . . 72
vide . . . 9, 12, 46, 50
valide si . . . 12, 53, 58
`Value` . . . 109
variable locale . . . 119
`Variant` . . . 113
VBA . . . 100, 122
véracité . . . 95
verrouillage . . . 76, 79
visibilité . . . 104
privée . . . 110
`Visible` . . . 109
`VraiFaux` . . . 95
vue . . . 35, 129
préalable . . . 43
sans clé . . . 36

Z

zone
état . . . 84
de liste . . . 67
à sélection multiple . . . 134
de texte . . . 66
verrouillée . . . 67